

Free Software, Free Society

Selected Essays of Richard M. Stallman
Second Edition



GNU Press

By Richard M. Stallman
Foreword by Lawrence Lessig

Richard Stallman is the prophet of the free software movement. He understood the dangers of software patents years ago. Now that this has become a crucial issue in the world, buy this book and read what he said.

—**Tim Berners-Lee**, inventor of the World Wide Web

Richard Stallman is the philosopher king of software. He single-handedly ignited what has become a world-wide movement to create software that is Free, with a capital F. He has toiled for years at a project that many once considered a fool's errand, and now that is widely seen as "inevitable."

—**Simon L. Garfinkel**, computer science author and columnist

By his hugely successful efforts to establish the idea of "Free Software," Stallman has made a massive contribution to the human condition. His contribution combines elements that have technical, social,

political, and economic consequences.

—**Gerald Jay Sussman**, Matsushita Professor of Electrical Engineering, MIT

RMS is the leading philosopher of software. You may dislike some of his attitudes, but you cannot avoid his ideas. This slim volume will make those ideas readily accessible to those who are confused by the buzzwords of rampant commercialism. This book needs to be widely circulated and widely read.

—**Peter Salus**, computer science writer, book reviewer, and UNIX historian

Richard is the leading force of the free software movement. This book is very important to spread the key concepts of free software world-wide, so everyone can understand it. Free software gives people freedom to use their creativity.

—**Masayuki Ida**, professor, Graduate School of International Management, Aoyama Gakuin University

Free Software, Free Society

Selected Essays of Richard M. Stallman
Second Edition

Richard M. Stallman

This is the second edition of *Free Software, Free Society: Selected Essays of Richard M. Stallman*.

Free Software Foundation
51 Franklin Street, Fifth Floor
Boston, MA 02110-1335

Copyright © 2002, 2010 Free Software Foundation,
Inc.

Verbatim copying and distribution of this entire book are permitted worldwide, without royalty, in any medium, provided this notice is preserved. Permission is granted to copy and distribute

translations of this book from the original English into another language provided the translation has been approved by the Free Software Foundation and the copyright notice and this permission notice are preserved on all copies.

ISBN 978-0-9831592-0-9

Cover design by Rob Myers.

Cover photograph by Peter Hinely.

Contents

Foreword

Preface to the Second Edition

I The GNU Project and Free Software

1 The Free Software Definition

2 The GNU Project

3 The Initial Announcement of the

GNU Operating System

4 The GNU Manifesto

5 Why Software Should Not Have Owners

6 Why Software Should Be Free

7 Why Schools Should Exclusively Use

Free Software

- 8 [Releasing Free Software If You Work at a University](#)
- 9 [Why Free Software Needs Free Documentation](#)
- 10 [Selling Free Software](#)
- 11 [The Free Software Song](#)
- II [What's in a Name?](#)
- 12 [What's in a Name?](#)
- 13 [Categories of Free and Nonfree Software](#)
- 14 [Why Open Source Misses the Point of Free Software](#)
- 15 [Did You Say "Intellectual Property"? It's a Seductive Mirage](#)
- 16 [Words to Avoid \(or Use with Care\) Because They Are Loaded or Confusing](#)

[They Are Loaded or Confusing](#)

III [Copyright, Copyleft](#)

17 [The Right to Read: A Dystopian Short Story](#)

18 [Misinterpreting Copyright—A Series of Errors](#)

19 [Science Must Push Copyright Aside](#)

20 [Freedom—or Copyright](#)

21 [What Is Copyleft?](#)

22 [Copyleft: Pragmatic Idealism](#)

IV [Software Patents: Danger to Programmers](#)

23 [Anatomy of a Trivial Patent](#)

24 [Software Patents and Literary Patents](#)

25 [The Danger of Software Patents](#)

26 [Microsoft's New Monopoly](#)

V [The Licenses](#)

27 [Introduction to the Licenses](#)

28 [The GNU General Public License](#)

29 [Why Upgrade to GPLv3](#)

30 [The GNU Lesser General Public License](#)

31 [GNU Free Documentation License](#)

VI [Traps and Challenges](#)

32 [Can You Trust Your Computer?](#)

33 [Who Does That Server Really Serve?](#)

34 [Free but Shackled: The Java Trap](#)

35 [The JavaScript Trap](#)

36 [The X Window System Trap](#)

37 [The Problem Is Software Controlled by](#)

Its Developer

38 We Can Put an End to Word Attachments

39 Thank You, Larry McVoy

VII An Assessment and a Look Ahead

40 Computing “Progress”: Good and Bad

41 Avoiding Ruinous Compromises

42 Overcoming Social Inertia

43 Freedom or Power?

VIII Appendices

A A Note on Software

B Translations of the Term “Free Software”

Foreword

Copyright © 2002 Free Software Foundation, Inc.

This foreword was originally published, in 2002, as the introduction to the first edition. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Every generation has its philosopher—a writer or an artist who captures the imagination of a time. Sometimes these philosophers are recognized as such; often it takes generations before the connection is made real. But recognized or not, a time gets marked by the people who speak its ideals, whether in the whisper of a poem, or the blast of a political movement.

Our generation has a philosopher. He is not an artist, or a professional writer. He is a programmer. Richard Stallman began his work in the labs of MIT as a

Stallman began his work in the late 1970s, as a programmer and architect building operating system software. He has built his career on a stage of public life, as a programmer and an architect founding a movement for freedom in a world increasingly defined by “code.”

“Code” is the technology that makes computers run. Whether inscribed in software or burned in hardware, it is the collection of instructions, first written in words, that directs the functionality of machines. These machines—computers—increasingly define and control our life. They determine how phones connect, and what runs on TV. They decide whether video can be streamed across a broadband link to a computer. They control what a computer reports back to its manufacturer. These machines run us. Code runs these machines.

What control should we have over this code? What understanding? What freedom should there be to match the control it enables? What power?

These questions have been the challenge of Stallman’s life. Through his works and his words, he has pushed us to see the importance of keeping code “free.”

Not free in the sense that code writers don't get paid, but free in the sense that the control coders build be transparent to all, and that anyone have the right to take that control, and modify it as he or she sees fit. This is "free software"; "free software" is one answer to a world built in code.

"Free." Stallman laments the ambiguity in his own term. There's nothing to lament. Puzzles force people to think, and this term "free" does this puzzling work quite well. To modern American ears, "free software" sounds utopian, impossible. Nothing, not even lunch, is free. How could the most important words running the most critical machines running the world be "free." How could a sane society aspire to such an ideal?

Yet the odd clink of the word "free" is a function of us, not of the term. "Free" has different senses, only one of which refers to "price." A much more fundamental sense of "free" is the "free," Stallman says, in the term "free speech," or perhaps better in the term "free labor." Not free as in costless, but free as in limited in its control by others. Free software is control that is transparent, and open to change, just as free love, or the love of a

and open to change, just as free laws, or the laws of a “free society,” are free when they make their control knowable, and open to change. The aim of Stallman’s “free software movement” is to make as much code as it can transparent, and subject to change, by rendering it “free.”

The mechanism of this rendering is an extraordinarily clever device called “copyleft” implemented through a license called GPL. Using the power of copyright law, “free software” not only assures that it remains open, and subject to change, but that other software that takes and uses “free software” (and that technically counts as a “derivative”) must also itself be free. If you use and adapt a free software program, and then release that adapted version to the public, the released version must be as free as the version it was adapted from. It must, or the law of copyright will be violated.

“Free software,” like free societies, has its enemies. Microsoft has waged a war against the GPL, warning whoever will listen that the GPL is a “dangerous” license. The dangers it names, however, are largely illusory. Others object to the “coercion” in GPL’s insistence that

modified versions are also free. But a condition is not coercion. If it is not coercion for Microsoft to refuse to permit users to distribute modified versions of its product Office without paying it (presumably) millions, then it is not coercion when the GPL insists that modified versions of free software be free too.

And then there are those who call Stallman's message too extreme. But extreme it is not. Indeed, in an obvious sense, Stallman's work is a simple translation of the freedoms that our tradition crafted in the world before code. "Free software" would assure that the world governed by code is as "free" as our tradition that built the world before code.

For example: A "free society" is regulated by law. But there are limits that any free society places on this regulation through law: No society that kept its laws secret could ever be called free. No government that hid its regulations from the regulated could ever stand in our tradition. Law controls. But it does so justly only when visibly. And law is visible only when its terms are knowable and controllable by those it regulates, or by the

agents of those it regulates (lawyers, legislatures).

This condition on law extends beyond the work of a legislature. Think about the practice of law in American courts. Lawyers are hired by their clients to advance their clients' interests. Sometimes that interest is advanced through litigation. In the course of this litigation, lawyers write briefs. These briefs in turn affect opinions written by judges. These opinions decide who wins a particular case, or whether a certain law can stand consistently with a constitution.

All the material in this process is free in the sense that Stallman means. Legal briefs are open and free for others to use. The arguments are transparent (which is different from saying they are good) and the reasoning can be taken without the permission of the original lawyers. The opinions they produce can be quoted in later briefs. They can be copied and integrated into another brief or opinion. The “source code” for American law is by design, and by principle, open and free for anyone to take. And take lawyers do—for it is a measure of a great brief that it achieves its creativity through the reuse of what happened before. The source is free: creativity and

an economy is built upon it.

This economy of free code (and here I mean free legal code) doesn't starve lawyers. Law firms have enough incentive to produce great briefs even though the stuff they build can be taken and copied by anyone else. The lawyer is a craftsman; his or her product is public. Yet the crafting is not charity. Lawyers get paid; the public doesn't demand such work without price. Instead this economy flourishes, with later work added to the earlier.

We could imagine a legal practice that was different—briefs and arguments that were kept secret; rulings that announced a result but not the reasoning. Laws that were kept by the police but published to no one else. Regulation that operated without explaining its rule.

We could imagine this society, but we could not imagine calling it “free.” Whether or not the incentives in such a society would be better or more efficiently allocated, such a society could not be known as free. The ideals of freedom, of life within a free society,

demand more than efficient application. Instead, openness and transparency are the constraints within which a legal system gets built, not options to be added if convenient to the leaders. Life governed by software code should be no less.

Code writing is not litigation. It is better, richer, more productive. But the law is an obvious instance of how creativity and incentives do not depend upon perfect control over the products created. Like jazz, or novels, or architecture, the law gets built upon the work that went before. This adding and changing is what creativity always is. And a free society is one that assures that its most important resources remain free in just this sense.

This book collects the writing of Richard Stallman in a manner that will make its subtlety and power clear. The essays span a wide range, from copyright to the history of the free software movement. They include many arguments not well known, and among these, an especially insightful account of the changed circumstances that render copyright in the digital world suspect. They will serve as a resource for those who seek to understand the thought of this most powerful man—

powerful in his ideas, his passion, and his integrity, even if powerless in every other way. They will inspire others who would take these ideas, and build upon them.

I don't know Stallman well. I know him well enough to know he is a hard man to like. He is driven, often impatient. His anger can flare at friend as easily as foe. He is uncompromising and persistent; patient in both.

Yet when our world finally comes to understand the power and danger of code—when it finally sees that code, like laws, or like government, must be transparent to be free—then we will look back at this uncompromising and persistent programmer and recognize the vision he has fought to make real: the vision of a world where freedom and knowledge survives the compiler. And we will come to see that no man, through his deeds or words, has done as much to make possible the freedom that this next society could have.

We have not earned that freedom yet. We may well fail in securing it. But whether we succeed or fail, in these essays is a picture of what that freedom could be. And in

the life that produced these words and works, there is inspiration for anyone who would, like Stallman, fight to create this freedom.

LAWRENCE LESSIG

Lawrence Lessig is a Professor of Law at Harvard Law School, the director of the Edmond J. Safra Foundation Center for Ethics, and the founder of Stanford Law School's Center for Internet and Society. For much of his career, he focused his work on law and technology, especially as it affects copyright. He is the author of numerous books and has served as a board member of many organizations, including the Free Software Foundation.

Preface to the Second Edition

The second edition of *Free Software, Free Society* holds updated versions of most of the essays from the first edition, as well as many new essays published since the first edition.

The essays about software patents are now in one section and those about copyright in another, to set an example of not grouping together these two laws, whose workings and effects on software are totally different.

Another section presents the GNU licenses, with a new introduction written with Brett Smith giving their history and the motives for each of them. One of the essays explains why software projects should upgrade to version 3 of the GNU General Public License.

There is now a section on issues of terminology, since the way we describe an issue affects how people think about it.

The last two sections describe some of the trans free

software developers and users face—new ways to lose your freedom, and how to avoid them.

We have also added an index, to complement the appendix on software.

We would like to thank Jeanne Rasata for managing the project, editing the book, formatting the text, and creating the index. Thanks also to Karl Berry for technical assistance with Texinfo, Brett Smith for all other technical help and for valuable feedback, and Rob Myers for formatting the cover.

Part I
The GNU Project and
Free Software

Chapter 1

The Free Software Definition

Copyright © 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2004, 2005, 2006, 2007, 2009, 2010 Free Software Foundation, Inc.

The free software definition was first published in 1996, on <http://gnu.org>. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

We maintain this free software definition to show clearly what must be true about a particular software program for it to be considered free software. From time to time we revise this definition to clarify it. If you would like to review the changes we've made, please see the History section, following the definition, at <http://gnu.org/philosophy/free-sw.html>.

“Free software” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech,” not as in “free beer.”

Free software is a matter of the users’ freedom to run, copy, distribute, study, change and improve the software. More precisely, it means that the program’s users have the four essential freedoms:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and change it to make it do what you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

A program is free software if users have all of these

freedoms. Thus, you should be free to redistribute copies, either with or without modifications, either gratis or charging a fee for distribution, to anyone anywhere. Being free to do these things means (among other things) that you do not have to ask or pay for permission to do so.

You should also have the freedom to make modifications and use them privately in your own work or play, without even mentioning that they exist. If you do publish your changes, you should not be required to notify anyone in particular, or in any particular way.

The freedom to run the program means the freedom for any kind of person or organization to use it on any kind of computer system, for any kind of overall job and purpose, without being required to communicate about it with the developer or any other specific entity. In this freedom, it is the *user's* purpose that matters, not the *developer's* purpose; you as a user are free to run the program for your purposes, and if you distribute it to someone else, she is then free to run it for her purposes, but you are not entitled to impose your purposes on her.

The freedom to redistribute copies must include binary or executable forms of the program, as well as source code, for both modified and unmodified versions. (Distributing programs in runnable form is necessary for conveniently installable free operating systems.) It is OK if there is no way to produce a binary or executable form for a certain program (since some languages don't support that feature), but you must have the freedom to redistribute such forms should you find or develop a way to make them.

In order for freedoms 1 and 3 (the freedom to make changes and the freedom to publish improved versions) to be meaningful, you must have access to the source code of the program. Therefore, accessibility of source code is a necessary condition for free software. Obfuscated “source code” is not real source code and does not count as source code.

Freedom 1 includes the freedom to use your changed version in place of the original. If the program is delivered in a product designed to run someone else's modified versions but refuse to run yours—a practice known as

“tivoization” or (in its practitioners’ perverse terminology) as “secure boot”—freedom 1 becomes a theoretical fiction rather than a practical freedom. This is not sufficient. In other words, these binaries are not free software even if the source code they are compiled from is free.

One important way to modify a program is by merging in available free subroutines and modules. If the program’s license says that you cannot merge in a suitably licensed existing module—for instance, if it requires you to be the copyright holder of any code you add—then the license is too restrictive to qualify as free.

Freedom 3 includes the freedom to release your modified versions as free software. A free license may also permit other ways of releasing them; in other words, it does not have to be a copyleft license. However, a license that requires modified versions to be nonfree does not qualify as a free license.

In order for these freedoms to be real, they must be permanent and irrevocable as long as you do nothing

wrong; if the developer of the software has the power to revoke the license, or retroactively change its terms, without your doing anything wrong to give cause, the software is not free.

However, certain kinds of rules about the manner of distributing free software are acceptable, when they don't conflict with the central freedoms. For example, copyleft (very simply stated) is the rule that when redistributing the program, you cannot add restrictions to deny other people the central freedoms. This rule does not conflict with the central freedoms; rather it protects them.

“Free software” does not mean “noncommercial.” A free program must be available for commercial use, commercial development, and commercial distribution. Commercial development of free software is no longer unusual; such free commercial software is very important. You may have paid money to get copies of free software, or you may have obtained copies at no charge. But regardless of how you got your copies, you always have the freedom to copy and change the software, even to sell copies.

Whether a change constitutes an improvement is a subjective matter. If your modifications are limited, in substance, to changes that someone else considers an improvement, that is not freedom.

However, rules about how to package a modified version are acceptable, if they don't substantively limit your freedom to release modified versions, or your freedom to make and use modified versions privately. Thus, it is acceptable for the license to require that you change the name of the modified version, remove a logo, or identify your modifications as yours. As long as these requirements are not so burdensome that they effectively hamper you from releasing your changes, they are acceptable; you're already making other changes to the program, so you won't have trouble making a few more.

Rules that "if you make your version available in this way, you must make it available in that way also" can be acceptable too, on the same condition. An example of such an acceptable rule is one saying that if you have distributed a modified version and a previous developer asks for a copy of it, you must send one. (Note that such

asks for a copy of it, you must send one. (Note that such a rule still leaves you the choice of whether to distribute your version at all.) Rules that require release of source code to the users for versions that you put into public use are also acceptable.

In the GNU Project, we use copyleft to protect these freedoms legally for everyone. But noncopylefted free software also exists. We believe there are important reasons why it is better to use copyleft, but if your program is noncopylefted free software, it is still basically ethical. (See “Categories of Free and Nonfree Software” ([119](#)) for a description of how “free software,” “copylefted software” and other categories of software relate to each other.)

Sometimes government export control regulations and trade sanctions can constrain your freedom to distribute copies of programs internationally. Software developers do not have the power to eliminate or override these restrictions, but what they can and must do is refuse to impose them as conditions of use of the program. In this way, the restrictions will not affect activities and people outside the jurisdictions of these

governments. Thus, free software licenses must not require obedience to any export regulations as a condition of any of the essential freedoms.

Most free software licenses are based on copyright, and there are limits on what kinds of requirements can be imposed through copyright. If a copyright-based license respects freedom in the ways described above, it is unlikely to have some other sort of problem that we never anticipated (though this does happen occasionally). However, some free software licenses are based on contracts, and contracts can impose a much larger range of possible restrictions. That means there are many possible ways such a license could be unacceptably restrictive and nonfree.

We can't possibly list all the ways that might happen. If a contract-based license restricts the user in an unusual way that copyright-based licenses cannot, and which isn't mentioned here as legitimate, we will have to think about it, and we will probably conclude it is nonfree.

When talking about free software, it is best to avoid using terms like “copyleft” or “free-free” because these

using terms like give away or for free, because those terms imply that the issue is about price, not freedom. Some common terms such as “piracy” embody opinions we hope you won’t endorse. See “Words to Avoid (or Use with Care)” ([143](#)) for a discussion of these terms. We also have a list of proper translations of “free software” into various languages ([393](#)).

Finally, note that criteria such as those stated in this free software definition require careful thought for their interpretation. To decide whether a specific software license qualifies as a free software license, we judge it based on these criteria to determine whether it fits their spirit as well as the precise words. If a license includes unconscionable restrictions, we reject it, even if we did not anticipate the issue in these criteria. Sometimes a license requirement raises an issue that calls for extensive thought, including discussions with a lawyer, before we can decide if the requirement is acceptable. When we reach a conclusion about a new issue, we often update these criteria to make it easier to see why certain licenses do or don’t qualify.

If you are interested in whether a specific license

qualifies as a free software license, see our list of licenses, at <http://gnu.org/licenses/license-list.html>. If the license you are concerned with is not listed there, you can ask us about it by sending us email at licensing@gnu.org.

If you are contemplating writing a new license, please contact the Free Software Foundation first by writing to that address. The proliferation of different free software licenses means increased work for users in understanding the licenses; we may be able to help you find an existing free software license that meets your needs.

If that isn't possible, if you really need a new license, with our help you can ensure that the license really is a free software license and avoid various practical problems.

Beyond Software

Software manuals must be free, for the same reasons that software must be free, and because the manuals are in effect part of the software.

The same arguments also make sense for other kinds of works of practical use—that is to say, works that embody useful knowledge, such as educational works and reference works. Wikipedia is the best-known example.

Any kind of work *can* be free, and the definition of free software has been extended to a definition of free cultural works [\[1\]](#) applicable to any kind of works.

Endnotes

[1](#) See <http://freedomdefined.org>.

Chapter 2

The GNU Project

Copyright © 1998, 2001, 2002, 2005, 2006, 2007, 2008, 2010 Richard Stallman

The original version of this essay was published in *Open Sources: Voices from the Open Source Revolution*, by Chris DiBona and others (Sebastopol: O'Reilly Media, 1999), under the title “The GNU Operating System and the Free Software Movement.” This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

The First Software-Sharing Community

When I started working at the MIT Artificial Intelligence Lab in 1971, I became part of a software-sharing community that had existed for many years. Sharing of software was not limited to our particular community; it is

as old as computers, just as sharing of recipes is as old as cooking. But we did it more than most.

The AI Lab used a timesharing operating system called ITS (the Incompatible Timesharing System) that the lab's staff hackers [1] had designed and written in assembler language for the Digital PDP-10, one of the large computers of the era. As a member of this community, an AI Lab staff system hacker, my job was to improve this system.

We did not call our software “free software,” because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program.

The Collapse of the Community

The situation changed drastically in the early 1980s when Digital discontinued the PDP-10 series. Its architecture

Digital discontinued the PDP-10 series. Its architecture, elegant and powerful in the 60s, could not extend naturally to the larger address spaces that were becoming feasible in the 80s. This meant that nearly all of the programs composing ITS were obsolete.

The AI Lab hacker community had already collapsed, not long before. In 1981, the spin-off company Symbolics had hired away nearly all of the hackers from the AI Lab, and the depopulated community was unable to maintain itself. (The book *Hackers*, by Steve Levy, describes these events, as well as giving a clear picture of this community in its prime.) When the AI Lab bought a new PDP-10 in 1982, its administrators decided to use Digital's nonfree timesharing system instead of ITS.

The modern computers of the era, such as the VAX or the 68020, had their own operating systems, but none of them were free software: you had to sign a nondisclosure agreement even to get an executable copy.

This meant that the first step in using a computer was to promise not to help your neighbor. A cooperating

community was forbidden. The rule made by the owners of proprietary software was, “If you share with your neighbor, you are a pirate. If you want any changes, beg us to make them.”

The idea that the proprietary software social system—the system that says you are not allowed to share or change software—is antisocial, that it is unethical, that it is simply wrong, may come as a surprise to some readers. But what else could we say about a system based on dividing the public and keeping users helpless? Readers who find the idea surprising may have taken the proprietary software social system as a given, or judged it on the terms suggested by proprietary software businesses. Software publishers have worked long and hard to convince people that there is only one way to look at the issue.

When software publishers talk about “enforcing” their “rights” or “stopping piracy,” what they actually *say* is secondary. The real message of these statements is in the unstated assumptions they take for granted, which the public is asked to accept without examination. Let’s

therefore examine them.

One assumption is that software companies have an unquestionable natural right to own software and thus have power over all its users. (If this were a natural right, then no matter how much harm it does to the public, we could not object.) Interestingly, the US Constitution and legal tradition reject this view; copyright is not a natural right, but an artificial government-imposed monopoly that limits the users' natural right to copy.

Another unstated assumption is that the only important thing about software is what jobs it allows you to do—that we computer users should not care what kind of society we are allowed to have.

A third assumption is that we would have no usable software (or would never have a program to do this or that particular job) if we did not offer a company power over the users of the program. This assumption may have seemed plausible, before the free software movement demonstrated that we can make plenty of useful software without putting chains on it.

If we decline to accept these assumptions, and judge these issues based on ordinary commonsense morality while placing the users first, we arrive at very different conclusions. Computer users should be free to modify programs to fit their needs, and free to share software, because helping other people is the basis of society.

There is no room here for an extensive statement of the reasoning behind this conclusion, so I refer the reader to the article “Why Software Should Not Have Owners” ([55](#)).

A Stark Moral Choice

With my community gone, to continue as before was impossible. Instead, I faced a stark moral choice.

The easy choice was to join the proprietary software world, signing nondisclosure agreements and promising not to help my fellow hacker. Most likely I would also be developing software that was released under nondisclosure agreements, thus adding to the pressure on other people to betray their fellows too.

I could have made money this way, and perhaps amused myself writing code. But I knew that at the end of my career, I would look back on years of building walls to divide people, and feel I had spent my life making the world a worse place.

I had already experienced being on the receiving end of a nondisclosure agreement, when someone refused to give me and the MIT AI Lab the source code for the control program for our printer. (The lack of certain features in this program made use of the printer extremely frustrating.) So I could not tell myself that nondisclosure agreements were innocent. I was very angry when he refused to share with us; I could not turn around and do the same thing to everyone else.

Another choice, straightforward but unpleasant, was to leave the computer field. That way my skills would not be misused, but they would still be wasted. I would not be culpable for dividing and restricting computer users, but it would happen nonetheless.

So I looked for a way that a programmer could do

something for the good. I asked myself, was there a program or programs that I could write, so as to make a community possible once again?

The answer was clear: what was needed first was an operating system. That is the crucial software for starting to use a computer. With an operating system, you can do many things; without one, you cannot run the computer at all. With a free operating system, we could again have a community of cooperating hackers—and invite anyone to join. And anyone would be able to use a computer without starting out by conspiring to deprive his or her friends.

As an operating system developer, I had the right skills for this job. So even though I could not take success for granted, I realized that I was elected to do the job. I chose to make the system compatible with Unix so that it would be portable, and so that Unix users could easily switch to it. The name GNU was chosen, following a hacker tradition, as a recursive acronym for “GNU’s Not Unix.”

An operating system does not mean just a kernel,

barely enough to run other programs. In the 1970s, every operating system worthy of the name included command processors, assemblers, compilers, interpreters, debuggers, text editors, mailers, and much more. ITS had them, Multics had them, VMS had them, and Unix had them. The GNU operating system would include them too.

Later I heard these words, attributed to Hillel: [\[2\]](#)

If I am not for myself, who will be for me? If I am only for myself, what am I? If not now, when?

The decision to start the GNU Project was based on a similar spirit.

Free as in Freedom

The term “free software” is sometimes misunderstood—it has nothing to do with price. It is about freedom. Here, therefore, is the definition of free software.

A program is free software, for you, a particular user, if:

- You have the freedom to run the program as you wish, for any purpose.
- You have the freedom to modify the program to suit your needs. (To make this freedom effective in practice, you must have access to the source code, since making changes in a program without having the source code is exceedingly difficult.)
- You have the freedom to redistribute copies, either gratis or for a fee.
- You have the freedom to distribute modified versions of the program, so that the community can benefit from your improvements.

Since “free” refers to freedom, not to price, there is no contradiction between selling copies and free software. In fact, the freedom to sell copies is crucial: collections of free software sold on CD-ROMs are important for the community, and selling them is an important way to raise funds for free software development. Therefore, a program which people are not free to include on these collections is not free software.

Because of the ambiguity of “free,” people have long

Because of the ambiguity of free, people have long looked for alternatives, but no one has found a better term. The English language has more words and nuances than any other, but it lacks a simple, unambiguous, word that means “free,” as in freedom—“unfettered” being the word that comes closest in meaning. Such alternatives as “liberated,” “freedom,” and “open” have either the wrong meaning or some other disadvantage.

GNU Software and the GNU System

Developing a whole system is a very large project. To bring it into reach, I decided to adapt and use existing pieces of free software wherever that was possible. For example, I decided at the very beginning to use T_EX as the principal text formatter; a few years later, I decided to use the X Window System rather than writing another window system for GNU.

Because of this decision, the GNU system is not the same as the collection of all GNU software. The GNU system includes programs that are not GNU software, programs that were developed by other people and projects for their own purposes but which we can use

project for their own purposes, but which we can use because they are free software.

Commencing the Project

In January 1984 I quit my job at MIT and began writing GNU software. Leaving MIT was necessary so that MIT would not be able to interfere with distributing GNU as free software. If I had remained on the staff, MIT could have claimed to own the work, and could have imposed their own distribution terms, or even turned the work into a proprietary software package. I had no intention of doing a large amount of work only to see it become useless for its intended purpose: creating a new software-sharing community.

However, Professor Winston, then the head of the MIT AI Lab, kindly invited me to keep using the lab's facilities.

The First Steps

Shortly before beginning the GNU Project, I heard about the Free University Compiler Kit, also known as VUCK.

(The Dutch word for “free” is written with a *v*.) This was a compiler designed to handle multiple languages, including C and Pascal, and to support multiple target machines. I wrote to its author asking if GNU could use it.

He responded derisively, stating that the university was free but the compiler was not. I therefore decided that my first program for the GNU Project would be a multilanguage, multiplatform compiler.

Hoping to avoid the need to write the whole compiler myself, I obtained the source code for the Pastel compiler, which was a multiplatform compiler developed at Lawrence Livermore Lab. It supported, and was written in, an extended version of Pascal, designed to be a system-programming language. I added a C front end, and began porting it to the Motorola 68000 computer. But I had to give that up when I discovered that the compiler needed many megabytes of stack space, while the available 68000 Unix system would only allow 64k.

I then realized that the Pastel compiler functioned by parsing the entire input file into a syntax tree, converting

parsing the entire input file into a syntax tree, converting the whole syntax tree into a chain of “instructions,” and then generating the whole output file, without ever freeing any storage. At this point, I concluded I would have to write a new compiler from scratch. That new compiler is now known as GCC; none of the Pastel compiler is used in it, but I managed to adapt and use the C front end that I had written. But that was some years later; first, I worked on GNU Emacs.

GNU Emacs

I began work on GNU Emacs in September 1984, and in early 1985 it was beginning to be usable. This enabled me to begin using Unix systems to do editing; having no interest in learning to use vi or ed, I had done my editing on other kinds of machines until then.

At this point, people began wanting to use GNU Emacs, which raised the question of how to distribute it. Of course, I put it on the anonymous ftp server on the MIT computer that I used. (This computer, `prep.ai.mit.edu`, thus became the principal GNU ftp distribution site; when it was decommissioned a few

years later, we transferred the name to our new ftp server.) But at that time, many of the interested people were not on the Internet and could not get a copy by ftp. So the question was, what would I say to them?

I could have said, “Find a friend who is on the net and who will make a copy for you.” Or I could have done what I did with the original PDP-10 Emacs: tell them, “Mail me a tape and a SASE (self-addressed stamped envelope), and I will mail it back with Emacs on it.” But I had no job, and I was looking for ways to make money from free software. So I announced that I would mail a tape to whoever wanted one, for a fee of \$150. In this way, I started a free software distribution business, the precursor of the companies that today distribute entire Linux-based GNU systems.

Is a Program Free for Every User?

If a program is free software when it leaves the hands of its author, this does not necessarily mean it will be free software for everyone who has a copy of it. For example, public domain software (software that is not

copyrighted) is free software; but anyone can make a proprietary modified version of it. Likewise, many free programs are copyrighted but distributed under simple permissive licenses which allow proprietary modified versions.

The paradigmatic example of this problem is the X Window System. Developed at MIT, and released as free software with a permissive license, it was soon adopted by various computer companies. They added X to their proprietary Unix systems, in binary form only, and covered by the same nondisclosure agreement. These copies of X were no more free software than Unix was.

The developers of the X Window System did not consider this a problem—they expected and intended this to happen. Their goal was not freedom, just “success,” defined as “having many users.” They did not care whether these users had freedom, only about having many of them.

This led to a paradoxical situation where two different ways of counting the amount of freedom gave

different ways of counting the amount of freedom gave different answers to the question, “Is this program free?” If you judged based on the freedom provided by the distribution terms of the MIT release, you would say that X was free software. But if you measured the freedom of the average user of X, you would have to say it was proprietary software. Most X users were running the proprietary versions that came with Unix systems, not the free version.

Copyleft and the GNU GPL

The goal of GNU was to give users freedom, not just to be popular. So we needed to use distribution terms that would prevent GNU software from being turned into proprietary software. The method we use is called “copyleft.” [\[3\]](#)

Copyleft uses copyright law, but flips it over to serve the opposite of its usual purpose: instead of a means for restricting a program, it becomes a means for keeping the program free.

The central idea of copyleft is that we give everyone

permission to run the program, copy the program, modify the program, and distribute modified versions—but not permission to add restrictions of their own. Thus, the crucial freedoms that define “free software” are guaranteed to everyone who has a copy; they become inalienable rights.

For an effective copyleft, modified versions must also be free. This ensures that work based on ours becomes available to our community if it is published. When programmers who have jobs as programmers volunteer to improve GNU software, it is copyleft that prevents their employers from saying, “You can’t share those changes, because we are going to use them to make our proprietary version of the program.”

The requirement that changes must be free is essential if we want to ensure freedom for every user of the program. The companies that privatized the X Window System usually made some changes to port it to their systems and hardware. These changes were small compared with the great extent of X, but they were not trivial. If making changes were an excuse to deny the users freedom, it would be easy for anyone to take

users freedom, it would be easy for anyone to take advantage of the excuse.

A related issue concerns combining a free program with nonfree code. Such a combination would inevitably be nonfree; whichever freedoms are lacking for the nonfree part would be lacking for the whole as well. To permit such combinations would open a hole big enough to sink a ship. Therefore, a crucial requirement for copyleft is to plug this hole: anything added to or combined with a copylefted program must be such that the larger combined version is also free and copylefted.

The specific implementation of copyleft that we use for most GNU software is the GNU General Public License, or GNU GPL for short. We have other kinds of copyleft that are used in specific circumstances. GNU manuals are copylefted also, but use a much simpler kind of copyleft, because the complexity of the GNU GPL is not necessary for manuals. [\[4\]](#)

The Free Software Foundation

As interest in using Emacs was growing, other people

became involved in the GNU Project, and we decided that it was time to seek funding once again. So in 1985 we created the Free Software Foundation (FSF), a tax-exempt charity for free software development. The FSF also took over the Emacs tape distribution business; later it extended this by adding other free software (both GNU and non-GNU) to the tape, and by selling free manuals as well.

Most of the FSF's income used to come from sales of copies of free software and of other related services (CD-ROMs of source code, CD-ROMs with binaries, nicely printed manuals, all with the freedom to redistribute and modify), and Deluxe Distributions (distributions for which we built the whole collection of software for the customer's choice of platform). Today the FSF still sells manuals and other gear, but it gets the bulk of its funding from members' dues. You can join the FSF at <http://fsf.org/join>.

Free Software Foundation employees have written and maintained a number of GNU software packages. Two notable ones are the C library and the shell. The

GNU C library is what every program running on a GNU/Linux system uses to communicate with Linux. It was developed by a member of the Free Software Foundation staff, Roland McGrath. The shell used on most GNU/Linux systems is BASH, the Bourne Again Shell, [\[5\]](#) which was developed by FSF employee Brian Fox.

We funded development of these programs because the GNU Project was not just about tools or a development environment. Our goal was a complete operating system, and these programs were needed for that goal.

Free Software Support

The free software philosophy rejects a specific widespread business practice, but it is not against business. When businesses respect the users' freedom, we wish them success.

Selling copies of Emacs demonstrates one kind of free software business. When the FSF took over that business, I needed another way to make a living. I found

it in selling services relating to the free software I had developed. This included teaching, for subjects such as how to program GNU Emacs and how to customize GCC, and software development, mostly porting GCC to new platforms.

Today each of these kinds of free software business is practiced by a number of corporations. Some distribute free software collections on CD-ROM; others sell support at levels ranging from answering user questions, to fixing bugs, to adding major new features. We are even beginning to see free software companies based on launching new free software products.

Watch out, though—a number of companies that associate themselves with the term “open source” actually base their business on nonfree software that works with free software. These are not free software companies, they are proprietary software companies whose products tempt users away from freedom. They call these programs “value-added packages,” which shows the values they would like us to adopt: convenience above freedom. If we value freedom more,

we should call them “freedom-subtracted” packages.

Technical Goals

The principal goal of GNU is to be free software. Even if GNU had no technical advantage over Unix, it would have a social advantage, allowing users to cooperate, and an ethical advantage, respecting the user’s freedom.

But it was natural to apply the known standards of good practice to the work—for example, dynamically allocating data structures to avoid arbitrary fixed size limits, and handling all the possible 8-bit codes wherever that made sense.

In addition, we rejected the Unix focus on small memory size, by deciding not to support 16-bit machines (it was clear that 32-bit machines would be the norm by the time the GNU system was finished), and to make no effort to reduce memory usage unless it exceeded a megabyte. In programs for which handling very large files was not crucial, we encouraged programmers to read an entire input file into core, then scan its contents without having to worry about I/O

having to worry about I/O.

These decisions enabled many GNU programs to surpass their Unix counterparts in reliability and speed.

Donated Computers

As the GNU Project's reputation grew, people began offering to donate machines running Unix to the project. These were very useful, because the easiest way to develop components of GNU was to do it on a Unix system, and replace the components of that system one by one. But they raised an ethical issue: whether it was right for us to have a copy of Unix at all.

Unix was (and is) proprietary software, and the GNU Project's philosophy said that we should not use proprietary software. But, applying the same reasoning that leads to the conclusion that violence in self defense is justified, I concluded that it was legitimate to use a proprietary package when that was crucial for developing a free replacement that would help others stop using the proprietary package.

But, even if this was a justifiable evil, it was still an evil. Today we no longer have any copies of Unix, because we have replaced them with free operating systems. If we could not replace a machine's operating system with a free one, we replaced the machine instead.

The GNU Task List

As the GNU Project proceeded, and increasing numbers of system components were found or developed, eventually it became useful to make a list of the remaining gaps. We used it to recruit developers to write the missing pieces. This list became known as the GNU Task List. In addition to missing Unix components, we listed various other useful software and documentation projects that, we thought, a truly complete system ought to have.

Today, [\[6\]](#) hardly any Unix components are left in the GNU Task List—those jobs had been done, aside from a few inessential ones. But the list is full of projects that some might call “applications.” Any program that appeals to more than a narrow class of users would be a useful thing to add to an operating system.

useful thing to add to an operating system.

Even games are included in the task list—and have been since the beginning. Unix included games, so naturally GNU should too. But compatibility was not an issue for games, so we did not follow the list of games that Unix had. Instead, we listed a spectrum of different kinds of games that users might like.

The GNU Library GPL

The GNU C library uses a special kind of copyleft called the GNU Library General Public License, [\[7\]](#) which gives permission to link proprietary software with the library. Why make this exception?

It is not a matter of principle; there is no principle that says proprietary software products are entitled to include our code. (Why contribute to a project predicated on refusing to share with us?) Using the LGPL for the C library, or for any library, is a matter of strategy.

The C library does a generic job; every proprietary system or compiler comes with a C library. Therefore, to

make our C library available only to free software would not have given free software any advantage—it would only have discouraged use of our library.

One system is an exception to this: on the GNU system (and this includes GNU/Linux), the GNU C library is the only C library. So the distribution terms of the GNU C library determine whether it is possible to compile a proprietary program for the GNU system. There is no ethical reason to allow proprietary applications on the GNU system, but strategically it seems that disallowing them would do more to discourage use of the GNU system than to encourage development of free applications. That is why using the Library GPL is a good strategy for the C library.

For other libraries, the strategic decision needs to be considered on a case-by-case basis. When a library does a special job that can help write certain kinds of programs, then releasing it under the GPL, limiting it to free programs only, is a way of helping other free software developers, giving them an advantage against proprietary software.

Consider GNU Readline, a library that was developed to provide command-line editing for BASH. Readline is released under the ordinary GNU GPL, not the Library GPL. This probably does reduce the amount Readline is used, but that is no loss for us. Meanwhile, at least one useful application has been made free software specifically so it could use Readline, and that is a real gain for the community.

Proprietary software developers have the advantages money provides; free software developers need to make advantages for each other. I hope some day we will have a large collection of GPL-covered libraries that have no parallel available to proprietary software, providing useful modules to serve as building blocks in new free software, and adding up to a major advantage for further free software development.

Scratching an Itch?

Eric Raymond [\[8\]](#) says that “Every good work of software starts by scratching a developer’s personal itch.” [\[9\]](#) Maybe that happens sometimes, but many

essential pieces of GNU software were developed in order to have a complete free operating system. They come from a vision and a plan, not from impulse.

For example, we developed the GNU C library because a Unix-like system needs a C library, BASH because a Unix-like system needs a shell, and GNU tar because a Unix-like system needs a tar program. The same is true for my own programs—the GNU C compiler, GNU Emacs, GDB and GNU Make.

Some GNU programs were developed to cope with specific threats to our freedom. Thus, we developed *gzip* to replace the Compress program, which had been lost to the community because of the LZW patents. We found people to develop *LessTif*, and more recently started GNOME and Harmony, to address the problems caused by certain proprietary libraries (see below). We are developing the GNU Privacy Guard to replace popular nonfree encryption software, because users should not have to choose between privacy and freedom.

Of course, the people writing these programs became interested in the work, and many features were

became interested in the work, and many features were added to them by various people for the sake of their own needs and interests. But that is not why the programs exist.

Unexpected Developments

At the beginning of the GNU Project, I imagined that we would develop the whole GNU system, then release it as a whole. That is not how it happened.

Since each component of the GNU system was implemented on a Unix system, each component could run on Unix systems long before a complete GNU system existed. Some of these programs became popular, and users began extending them and porting them—to the various incompatible versions of Unix, and sometimes to other systems as well.

The process made these programs much more powerful, and attracted both funds and contributors to the GNU Project. But it probably also delayed completion of a minimal working system by several years, as GNU developers' time was put into maintaining

these ports and adding features to the existing components, rather than moving on to write one missing component after another.

The GNU Hurd

By 1990, the GNU system was almost complete; the only major missing component was the kernel. We had decided to implement our kernel as a collection of server processes running on top of Mach. Mach is a microkernel developed at Carnegie Mellon University and then at the University of Utah; the GNU Hurd is a collection of servers (i.e., a herd of GNUs) that run on top of Mach, and do the various jobs of the Unix kernel. The start of development was delayed as we waited for Mach to be released as free software, as had been promised.

One reason for choosing this design was to avoid what seemed to be the hardest part of the job: debugging a kernel program without a source-level debugger to do it with. This part of the job had been done already, in Mach, and we expected to debug the Hurd servers as

user programs, with GDB. But it took a long time to make that possible, and the multithreaded servers that send messages to each other have turned out to be very hard to debug. Making the Hurd work solidly has stretched on for many years.

Alix

The GNU kernel was not originally supposed to be called the Hurd. Its original name was Alix—named after the woman who was my sweetheart at the time. She, a Unix system administrator, had pointed out how her name would fit a common naming pattern for Unix system versions; as a joke, she told her friends, “Someone should name a kernel after me.” I said nothing, but decided to surprise her with a kernel named Alix.

It did not stay that way. Michael (now Thomas) Bushnell, the main developer of the kernel, preferred the name Hurd, and redefined Alix to refer to a certain part of the kernel—the part that would trap system calls and handle them by sending messages to Hurd servers.

Later, Alix and I broke up, and she changed her name; independently, the Hurd design was changed so that the C library would send messages directly to servers, and this made the Alix component disappear from the design.

But before these things happened, a friend of hers came across the name Alix in the Hurd source code, and mentioned it to her. So she did have the chance to find a kernel named after her.

Linux and GNU/Linux

The GNU Hurd is not suitable for production use, and we don't know if it ever will be. The capability-based design has problems that result directly from the flexibility of the design, and it is not clear solutions exist.

Fortunately, another kernel is available. In 1991, Linus Torvalds developed a Unix-compatible kernel and called it Linux. In 1992, he made Linux free software; combining Linux with the not-quite-complete GNU system resulted in a complete free operating system.

(Combining them was a substantial job in itself, of course.) It is due to Linux that we can actually run a version of the GNU system today.

We call this system version GNU/Linux, to express its composition as a combination of the GNU system with Linux as the kernel.

Challenges in Our Future

We have proved our ability to develop a broad spectrum of free software. This does not mean we are invincible and unstoppable. Several challenges make the future of free software uncertain; meeting them will require steadfast effort and endurance, sometimes lasting for years. It will require the kind of determination that people display when they value their freedom and will not let anyone take it away.

The following four sections discuss these challenges.

Secret Hardware

Hardware manufacturers increasingly tend to keep hardware specifications secret. This makes it difficult to write free drivers so that Linux and XFree86 can support new hardware. We have complete free systems today, but we will not have them tomorrow if we cannot support tomorrow's computers.

There are two ways to cope with this problem. Programmers can do reverse engineering to figure out how to support the hardware. The rest of us can choose the hardware that is supported by free software; as our numbers increase, secrecy of specifications will become a self-defeating policy.

Reverse engineering is a big job; will we have programmers with sufficient determination to undertake it? Yes—if we have built up a strong feeling that free software is a matter of principle, and nonfree drivers are intolerable. And will large numbers of us spend extra money, or even a little extra time, so we can use free drivers? Yes, if the determination to have freedom is widespread.

[2008 note: this issue extends to the BIOS as well.

There is a free BIOS, coreboot; the problem is getting specs for machines so that coreboot can support them.]

Nonfree Libraries

A nonfree library that runs on free operating systems acts as a trap for free software developers. The library's attractive features are the bait; if you use the library, you fall into the trap, because your program cannot usefully be part of a free operating system. (Strictly speaking, we could include your program, but it won't *run* with the library missing.) Even worse, if a program that uses the proprietary library becomes popular, it can lure other unsuspecting programmers into the trap.

The first instance of this problem was the Motif toolkit, back in the 80s. Although there were as yet no free operating systems, it was clear what problem Motif would cause for them later on. The GNU Project responded in two ways: by asking individual free software projects to support the free X Toolkit widgets as well as Motif, and by asking for someone to write a free replacement for Motif. The job took many years;

LessTif, developed by the Hungry Programmers, became powerful enough to support most Motif applications only in 1997.

Between 1996 and 1998, another nonfree GUI toolkit library, called Qt, was used in a substantial collection of free software, the desktop KDE.

Free GNU/Linux systems were unable to use KDE, because we could not use the library. However, some commercial distributors of GNU/Linux systems who were not strict about sticking with free software added KDE to their systems—producing a system with more capabilities, but less freedom. The KDE group was actively encouraging more programmers to use Qt, and millions of new “Linux users” had never been exposed to the idea that there was a problem in this. The situation appeared grim.

The free software community responded to the problem in two ways: GNOME and Harmony.

GNOME, the GNU Network Object Model Environment, is GNU’s desktop project. Started in 1997

by Miguel de Icaza, and developed with the support of Red Hat Software, GNOME set out to provide similar desktop facilities, but using free software exclusively. It has technical advantages as well, such as supporting a variety of languages, not just C++. But its main purpose was freedom: not to require the use of any nonfree software.

Harmony is a compatible replacement library, designed to make it possible to run KDE software without using Qt.

In November 1998, the developers of Qt announced a change of license which, when carried out, should make Qt free software. There is no way to be sure, but I think that this was partly due to the community's firm response to the problem that Qt posed when it was nonfree. (The new license is inconvenient and inequitable, so it remains desirable to avoid using Qt.)

[Subsequent note: in September 2000, Qt was rereleased under the GNU GPL, which essentially solved this problem.]

How will we respond to the next tempting nonfree library? Will the whole community understand the need to stay out of the trap? Or will many of us give up freedom for convenience, and produce a major problem? Our future depends on our philosophy.

Software Patents

The worst threat we face comes from software patents, which can put algorithms and features off limits to free software for up to 20 years. The LZW compression algorithm patents were applied for in 1983, and we still cannot release free software to produce proper compressed GIFs. [As of 2009 they have expired.] In 1998, a free program to produce MP3 compressed audio was removed from distribution under threat of a patent suit.

There are ways to cope with patents: we can search for evidence that a patent is invalid, and we can look for alternative ways to do a job. But each of these methods works only sometimes; when both fail, a patent may force all free software to lack some feature that users

want. What will we do when this happens?

Those of us who value free software for freedom's sake will stay with free software anyway. We will manage to get work done without the patented features. But those who value free software because they expect it to be technically superior are likely to call it a failure when a patent holds it back. Thus, while it is useful to talk about the practical effectiveness of the “bazaar” model of development, and the reliability and power of some free software, we must not stop there. We must talk about freedom and principle.

Free Documentation

The biggest deficiency in our free operating systems is not in the software—it is the lack of good free manuals that we can include in our systems. Documentation is an essential part of any software package; when an important free software package does not come with a good free manual, that is a major gap. We have many such gaps today.

Free documentation, like free software, is a matter of freedom, not price. The criterion for a free manual is pretty much the same as for free software: it is a matter of giving all users certain freedoms. Redistribution (including commercial sale) must be permitted, online and on paper, so that the manual can accompany every copy of the program.

Permission for modification is crucial too. As a general rule, I don't believe that it is essential for people to have permission to modify all sorts of articles and books. For example, I don't think you or I are obliged to give permission to modify articles like this one, which describe our actions and our views.

But there is a particular reason why the freedom to modify is crucial for documentation for free software. When people exercise their right to modify the software, and add or change its features, if they are conscientious they will change the manual, too—so they can provide accurate and usable documentation with the modified program. A nonfree manual, which does not allow programmers to be conscientious and finish the job, does not fill our community's needs.

Some kinds of limits on how modifications are done pose no problem. For example, requirements to preserve the original author's copyright notice, the distribution terms, or the list of authors, are OK. It is also no problem to require modified versions to include notice that they were modified, even to have entire sections that may not be deleted or changed, as long as these sections deal with nontechnical topics. These kinds of restrictions are not a problem because they don't stop the conscientious programmer from adapting the manual to fit the modified program. In other words, they don't block the free software community from making full use of the manual.

However, it must be possible to modify all the *technical* content of the manual, and then distribute the result in all the usual media, through all the usual channels; otherwise, the restrictions do obstruct the community, the manual is not free, and we need another manual.

Will free software developers have the awareness and determination to produce a full spectrum of free

manuals! Once again, our future depends on philosophy.

We Must Talk about Freedom

Estimates today are that there are ten million users of GNU/Linux systems such as Debian GNU/Linux and Red Hat “Linux.” Free software has developed such practical advantages that users are flocking to it for purely practical reasons.

The good consequences of this are evident: more interest in developing free software, more customers for free software businesses, and more ability to encourage companies to develop commercial free software instead of proprietary software products.

But interest in the software is growing faster than awareness of the philosophy it is based on, and this leads to trouble. Our ability to meet the challenges and threats described above depends on the will to stand firm for freedom. To make sure our community has this will, we need to spread the idea to the new users as they come into the community.

But we are failing to do so: the efforts to attract new users into our community are far outstripping the efforts to teach them the civics of our community. We need to do both, and we need to keep the two efforts in balance.

“Open Source”

Teaching new users about freedom became more difficult in 1998, when a part of the community decided to stop using the term “free software” and say “open source software” instead.

Some who favored this term aimed to avoid the confusion of “free” with “gratis”—a valid goal. Others, however, aimed to set aside the spirit of principle that had motivated the free software movement and the GNU Project, and to appeal instead to executives and business users, many of whom hold an ideology that places profit above freedom, above community, above principle. Thus, the rhetoric of “open source” focuses on the potential to make high-quality, powerful software, but shuns the ideas of freedom, community, and principle.

The “Linux” magazines are a clear example of this—they are filled with advertisements for proprietary software that works with GNU/Linux. When the next Motif or Qt appears, will these magazines warn programmers to stay away from it, or will they run ads for it?

The support of business can contribute to the community in many ways; all else being equal, it is useful. But winning their support by speaking even less about freedom and principle can be disastrous; it makes the previous imbalance between outreach and civics education even worse.

“Free software” and “open source” describe the same category of software, more or less, but say different things about the software, and about values. The GNU Project continues to use the term “free software,” to express the idea that freedom, not just technology, is important.

Try!

Yoda’s aphorism (“There is no ‘try’”) sounds neat, but it

doesn't work for me. I have done most of my work while anxious about whether I could do the job, and unsure that it would be enough to achieve the goal if I did. But I tried anyway, because there was no one but me between the enemy and my city. Surprising myself, I have sometimes succeeded.

Sometimes I failed; some of my cities have fallen. Then I found another threatened city, and got ready for another battle. Over time, I've learned to look for threats and put myself between them and my city, calling on other hackers to come and join me.

Nowadays, often I'm not the only one. It is a relief and a joy when I see a regiment of hackers digging in to hold the line, and I realize, this city may survive—for now. But the dangers are greater each year, and now Microsoft has explicitly targeted our community. We can't take the future of freedom for granted. Don't take it for granted! If you want to keep your freedom, you must be prepared to defend it.

Endnotes

1 The use of “hacker” to mean “security breaker” is a confusion on the part of the mass media. We hackers refuse to recognize that meaning, and continue using the word to mean someone who loves to program, someone who enjoys playful cleverness, or the combination of the two. See my article, “On Hacking” at <http://stallman.org/articles/on-hacking/html>.

2 As an Atheist, I don’t follow any religious leaders, but I sometimes find I admire something one of them has said.

3 In 1984 or 1985, Don Hopkins (a very imaginative fellow) mailed me a letter. On the envelope he had written several amusing sayings, including this one: “Copyleft—all rights reversed.” I used the word “copyleft” to name the distribution concept I was developing at the time.

4 We now use the GNU Free Documentation License ([293](#)) for documentation.

5 “Bourne Again Shell” is a play on the name “Bourne Shell,” which was the usual shell on Unix.

6 That was written in 1998. In 2009 we no longer maintain a long task list. The community develops free software so fast that we can’t

even keep track of it all. Instead, we have a list of High Priority Projects, a much shorter list of projects we really want to encourage people to write.

[7](#) This license is now called the GNU Lesser General Public License, to avoid giving the idea that all libraries ought to use it.

[8](#) Eric Raymond is a prominent open source advocate; see “Why Open Source Misses the Point” ([129](#)).

[9](#) Eric S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, rev. ed. (Sebastopol, Calif.: O’Reilly, 2001), p. 23.

Chapter 3

The Initial Announcement of the GNU Operating System

Copyright © 1983 Richard Stallman

This announcement is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

This is the original announcement of the GNU Project, posted by Richard Stallman on 27 September 1983.

The actual history of the GNU Project differs in many ways from this initial plan. For example, the beginning was delayed until January 1984. Several of the philosophical concepts of free software were not clarified until a few years later.

From mit-vax!mit-eddie!RMS@MIT-OZ
From: RMS%MIT-OZ@mit-eddie
Newsgroups: net.unix-wizards,net.usoft
Subject: new Unix implementation
Date: Tue, 27-Sep-83 12:35:59 EST
Organization: MIT AI Lab, Cambridge, MA
Free Unix!

Starting this Thanksgiving I am going to write a complete Unix-compatible software system called GNU (for Gnu's Not Unix), and give it away free [\[1\]](#) to everyone who can use it. Contributions of time, money, programs and equipment are greatly needed.

To begin with, GNU will be a kernel plus all the utilities needed to write and run C programs: editor, shell, C compiler, linker, assembler, and a few other things. After this we will add a text formatter, a YACC, an Empire game, a spreadsheet, and hundreds of other things. We hope to supply, eventually, everything useful that normally comes with a Unix system, and anything else useful, including on-line and hardcopy documentation.

GNU will be able to run Unix programs, but will not be identical to Unix. We will make all improvements that are convenient, based on our experience with other operating systems. In particular, we plan to have longer filenames, file version numbers, a crashproof file system, filename completion perhaps, terminal-independent display support,

and eventually a Lisp-based window system through which several Lisp programs and ordinary Unix programs can share a screen. Both C and Lisp will be available as system programming languages. We will have network software based on MIT's chaosnet protocol, far superior to UUCP. We may also have something compatible with UUCP.

Who Am I?

I am Richard Stallman, inventor of the original much-imitated EMACS editor, now at the Artificial Intelligence Lab at MIT. I have worked extensively on compilers, editors, debuggers, command interpreters, the Incompatible Timesharing System and the Lisp Machine operating system. I pioneered terminal-independent display support in ITS. In addition I have implemented one crashproof file system and two window systems for Lisp machines.

Why I Must Write GNU

I consider that the golden rule requires that if I like a program I must share it with other people who like it. I cannot in good conscience sign a nondisclosure agreement or a software license agreement.

So that I can continue to use computers without violating my principles, I have decided to put together a

sufficient body of free software so that I will be able to get along without any software that is not free.

How You Can Contribute

I am asking computer manufacturers for donations of machines and money. I'm asking individuals for donations of programs and work.

One computer manufacturer has already offered to provide a machine. But we could use more. One consequence you can expect if you donate machines is that GNU will run on them at an early date. The machine had better be able to operate in a residential area, and not require sophisticated cooling or power.

Individual programmers can contribute by writing a compatible duplicate of some Unix utility and giving it to me. For most projects, such part-time distributed work would be very hard to coordinate; the independently-written parts would not work together. But for the particular task of replacing Unix, this problem is absent. Most interface specifications are fixed by Unix compatibility. If each contribution works with the rest of Unix, it will probably work with the rest of GNU.

If I get donations of money, I may be able to hire a few people full or part time. The salary won't be high, but

I'm looking for people for whom knowing they are helping humanity is as important as money. I view this as a way of enabling dedicated people to devote their full energies to working on GNU by sparing them the need to make a living in another way.

For more information, contact me.

Arpanet mail:

RMS@MIT-MC.ARPA

Usenet:

...!mit-eddie!RMS@OZ ...!mit-vax!RMS@OZ

US Snail:

Richard Stallman

166 Prospect St

Cambridge, MA 02139

Endnotes

1 The wording here was careless. The intention was that nobody would have to pay for *permission* to use the GNU system. But the words don't make this clear, and people often interpret them as saying that copies of GNU should always be distributed at little or no charge. That was never the intent.

Chapter 4

The GNU Manifesto

Copyright © 1985, 1993, 2003, 2005, 2007, 2008, 2009, 2010 Free Software Foundation, Inc.

“The GNU Manifesto” was originally published in *Dr. Dobbs’s Journal*, vol. 10, n. 3 (March 1985). This footnoted version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

The GNU Manifesto was written by Richard Stallman at the beginning of the GNU Project, to ask for participation and support. For the first few years, it was updated in minor ways to account for developments, but now it seems best to leave it unchanged as most people have seen it.

Since that time, we have learned about certain common misunderstandings that different wording could help avoid. Footnotes added since 1993 help clarify these points.

For up-to-date information about the available GNU software, please see the information available on our web server, in particular our list of software. For how to contribute, see <http://gnu.org/help>.

What's GNU? Gnu's Not Unix!

GNU, which stands for Gnu's Not Unix, is the name for the complete Unix-compatible software system which I am writing so that I can give it away free to everyone who can use it. [1] Several other volunteers are helping me. Contributions of time, money, programs and equipment are greatly needed.

So far we have an Emacs text editor with Lisp for writing editor commands, a source level debugger, a yacc-compatible parser generator, a linker, and around 35 utilities. A shell (command interpreter) is nearly completed. A new portable optimizing C compiler has compiled itself and may be released this year. An initial kernel exists but many more features are needed to emulate Unix. When the kernel and compiler are finished, it will be possible to distribute a GNU system suitable for

program development. We will use \TeX as our text formatter, but an \nroff is being worked on. We will use the free, portable X window system as well. After this we will add a portable Common Lisp, an Empire game, a spreadsheet, and hundreds of other things, plus online documentation. We hope to supply, eventually, everything useful that normally comes with a Unix system, and more.

GNU will be able to run Unix programs, but will not be identical to Unix. We will make all improvements that are convenient, based on our experience with other operating systems. In particular, we plan to have longer file names, file version numbers, a crashproof file system, file name completion perhaps, terminal-independent display support, and perhaps eventually a Lisp-based window system through which several Lisp programs and ordinary Unix programs can share a screen. Both C and Lisp will be available as system programming languages. We will try to support UUCP, MIT Chaosnet, and Internet protocols for communication.

GNU is aimed initially at machines in the

68000/16000 class with virtual memory, because they are the easiest machines to make it run on. The extra effort to make it run on smaller machines will be left to someone who wants to use it on them.

To avoid horrible confusion, please pronounce the *g* in the word “GNU” when it is the name of this project.

Why I Must Write GNU

I consider that the Golden Rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will.

So that I can continue to use computers without dishonor, I have decided to put together a sufficient body

of free software so that I will be able to get along without any software that is not free. I have resigned from the AI Lab to deny MIT any legal excuse to prevent me from giving GNU away. [\[2\]](#)

Why GNU Will Be Compatible with Unix

Unix is not my ideal system, but it is not too bad. The essential features of Unix seem to be good ones, and I think I can fill in what Unix lacks without spoiling them. And a system compatible with Unix would be convenient for many other people to adopt.

How GNU Will Be Available

GNU is not in the public domain. Everyone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution. That is to say, proprietary modifications will not be allowed. I want to make sure that all versions of GNU remain free.

Why Many Other Programmers Want to

Help

I have found many other programmers who are excited about GNU and want to help. Many programmers are unhappy about the commercialization of system software. It may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat others as friends. The purchaser of software must choose between friendship and obeying the law. Naturally, many decide that friendship is more important. But those who believe in law often do not feel at ease with either choice. They become cynical and think that programming is just a way of making money.

By working on and using GNU rather than proprietary programs, we can be hospitable to everyone and obey the law. In addition, GNU serves as an example to inspire and a banner to rally others to join us in sharing. This can give us a feeling of harmony which is

impossible if we use software that is not free. For about half the programmers I talk to, this is an important happiness that money cannot replace.

How You Can Contribute

I am asking computer manufacturers for donations of machines and money. I'm asking individuals for donations of programs and work. [\[3\]](#)

One consequence you can expect if you donate machines is that GNU will run on them at an early date. The machines should be complete, ready to use systems, approved for use in a residential area, and not in need of sophisticated cooling or power.

I have found very many programmers eager to contribute part-time work for GNU. For most projects, such part-time distributed work would be very hard to coordinate; the independently written parts would not work together. But for the particular task of replacing Unix, this problem is absent. A complete Unix system contains hundreds of utility programs, each of which is documented separately. Most interface specifications are

documented separately. Most interface specifications are fixed by Unix compatibility. If each contributor can write a compatible replacement for a single Unix utility, and make it work properly in place of the original on a Unix system, then these utilities will work right when put together. Even allowing for Murphy to create a few unexpected problems, assembling these components will be a feasible task. (The kernel will require closer communication and will be worked on by a small, tight group.)

If I get donations of money, I may be able to hire a few people full or part time. The salary won't be high by programmers' standards, but I'm looking for people for whom building community spirit is as important as making money. I view this as a way of enabling dedicated people to devote their full energies to working on GNU by sparing them the need to make a living in another way.

Why All Computer Users Will Benefit

Once GNU is written, everyone will be able to obtain good system software free, just like air. [\[4\]](#)

This means much more than just saving everyone the price of a Unix license. It means that much wasteful duplication of system programming effort will be avoided. This effort can go instead into advancing the state of the art.

Complete system sources will be available to everyone. As a result, a user who needs changes in the system will always be free to make them himself, or hire any available programmer or company to make them for him. Users will no longer be at the mercy of one programmer or company which owns the sources and is in sole position to make changes.

Schools will be able to provide a much more educational environment by encouraging all students to study and improve the system code. Harvard's computer lab used to have the policy that no program could be installed on the system if its sources were not on public display, and upheld it by actually refusing to install certain programs. I was very much inspired by this.

Finally, the overhead of considering who owns the system software and what one is or is not entitled to do

system software and what one is or is not entitled to do with it will be lifted.

Arrangements to make people pay for using a program, including licensing of copies, always incur a tremendous cost to society through the cumbersome mechanisms necessary to figure out how much (that is, which programs) a person must pay for. And only a police state can force everyone to obey them. Consider a space station where air must be manufactured at great cost: charging each breather per liter of air may be fair, but wearing the metered gas mask all day and all night is intolerable even if everyone can afford to pay the air bill. And the TV cameras everywhere to see if you ever take the mask off are outrageous. It's better to support the air plant with a head tax and chuck the masks.

Copying all or parts of a program is as natural to a programmer as breathing, and as productive. It ought to be as free.

Some Easily Rebutted Objections to GNU's Goals

“Nobody will use it if it is free, because that means they can’t rely on any support.”

“You have to charge for the program to pay for providing the support.”

If people would rather pay for GNU plus service than get GNU free without service, a company to provide just service to people who have obtained GNU free ought to be profitable. [\[5\]](#)

We must distinguish between support in the form of real programming work and mere handholding. The former is something one cannot rely on from a software vendor. If your problem is not shared by enough people, the vendor will tell you to get lost.

If your business needs to be able to rely on support, the only way is to have all the necessary sources and tools. Then you can hire any available person to fix your problem; you are not at the mercy of any individual. With Unix, the price of sources puts this out of consideration for most businesses. With GNU this will be easy. It is still possible for there to be no available competent person

possible for there to be no available competent person, but this problem cannot be blamed on distribution arrangements. GNU does not eliminate all the world's problems, only some of them.

Meanwhile, the users who know nothing about computers need handholding: doing things for them which they could easily do themselves but don't know how.

Such services could be provided by companies that sell just handholding and repair service. If it is true that users would rather spend money and get a product with service, they will also be willing to buy the service having got the product free. The service companies will compete in quality and price; users will not be tied to any particular one. Meanwhile, those of us who don't need the service should be able to use the program without paying for the service.

“You cannot reach many people without advertising, and you must charge for the program to support that.” “It’s no use advertising a program people can get free.”

There are various forms of free or very cheap publicity that can be used to inform numbers of computer users about something like GNU. But it may be true that one can reach more microcomputer users with advertising. If this is really so, a business which advertises the service of copying and mailing GNU for a fee ought to be successful enough to pay for its advertising and more. This way, only the users who benefit from the advertising pay for it.

On the other hand, if many people get GNU from their friends, and such companies don't succeed, this will show that advertising was not really necessary to spread GNU. Why is it that free market advocates don't want to let the free market decide this? [\[6\]](#)

“My company needs a proprietary operating system to get a competitive edge.”

GNU will remove operating system software from the realm of competition. You will not be able to get an edge in this area, but neither will your competitors be able to get an edge over you. You and they will compete in other areas, while benefiting mutually in this one. If

your business is selling an operating system, you will not like GNU, but that's tough on you. If your business is something else, GNU can save you from being pushed into the expensive business of selling operating systems.

I would like to see GNU development supported by gifts from many manufacturers and users, reducing the cost to each. [\[7\]](#)

“Don’t programmers deserve a reward for their creativity?”

If anything deserves a reward, it is social contribution. Creativity can be a social contribution, but only in so far as society is free to use the results. If programmers deserve to be rewarded for creating innovative programs, by the same token they deserve to be punished if they restrict the use of these programs.

“Shouldn’t a programmer be able to ask for a reward for his creativity?”

There is nothing wrong with wanting pay for work, or seeking to maximize one’s income. as long as one does

looking to maximize one's income, as long as one does not use means that are destructive. But the means customary in the field of software today are based on destruction.

Extracting money from users of a program by restricting their use of it is destructive because the restrictions reduce the amount and the ways that the program can be used. This reduces the amount of wealth that humanity derives from the program. When there is a deliberate choice to restrict, the harmful consequences are deliberate destruction.

The reason a good citizen does not use such destructive means to become wealthier is that, if everyone did so, we would all become poorer from the mutual destructiveness. This is Kantian ethics; or, the Golden Rule. Since I do not like the consequences that result if everyone hoards information, I am required to consider it wrong for one to do so. Specifically, the desire to be rewarded for one's creativity does not justify depriving the world in general of all or part of that creativity.

“Won’t programmers starve?”

I could answer that nobody is forced to be a programmer. Most of us cannot manage to get any money for standing on the street and making faces. But we are not, as a result, condemned to spend our lives standing on the street making faces, and starving. We do something else.

But that is the wrong answer because it accepts the questioner’s implicit assumption: that without ownership of software, programmers cannot possibly be paid a cent. Supposedly it is all or nothing.

The real reason programmers will not starve is that it will still be possible for them to get paid for programming; just not paid as much as now.

Restricting copying is not the only basis for business in software. It is the most common basis [\[8\]](#) because it brings in the most money. If it were prohibited, or rejected by the customer, software business would move to other bases of organization which are now used less often. There are always numerous ways to organize any

kind of business.

Probably programming will not be as lucrative on the new basis as it is now. But that is not an argument against the change. It is not considered an injustice that sales clerks make the salaries that they now do. If programmers made the same, that would not be an injustice either. (In practice they would still make considerably more than that.)

“Don’t people have a right to control how their creativity is used?”

“Control over the use of one’s ideas” really constitutes control over other people’s lives; and it is usually used to make their lives more difficult.

People who have studied the issue of intellectual property rights [\[9\]](#) carefully (such as lawyers) say that there is no intrinsic right to intellectual property. The kinds of supposed intellectual property rights that the government recognizes were created by specific acts of legislation for specific purposes.

For example, the patent system was established to encourage inventors to disclose the details of their inventions. Its purpose was to help society rather than to help inventors. At the time, the life span of 17 years for a patent was short compared with the rate of advance of the state of the art. Since patents are an issue only among manufacturers, for whom the cost and effort of a license agreement are small compared with setting up production, the patents often do not do much harm. They do not obstruct most individuals who use patented products.

The idea of copyright did not exist in ancient times, when authors frequently copied other authors at length in works of nonfiction. This practice was useful, and is the only way many authors' works have survived even in part. The copyright system was created expressly for the purpose of encouraging authorship. In the domain for which it was invented—books, which could be copied economically only on a printing press—it did little harm, and did not obstruct most of the individuals who read the books.

All intellectual property rights are just licenses granted by society because it was thought, rightly or wrongly, that society as a whole would benefit by granting them. But in any particular situation, we have to ask: are we really better off granting such license? What kind of act are we licensing a person to do?

The case of programs today is very different from that of books a hundred years ago. The fact that the easiest way to copy a program is from one neighbor to another, the fact that a program has both source code and object code which are distinct, and the fact that a program is used rather than read and enjoyed, combine to create a situation in which a person who enforces a copyright is harming society as a whole both materially and spiritually; in which a person should not do so regardless of whether the law enables him to.

“Competition makes things get done better.”

The paradigm of competition is a race: by rewarding the winner, we encourage everyone to run faster. When capitalism really works this way, it does a good job; but its defenders are wrong in assuming it always works this

way. If the runners forget why the reward is offered and become intent on winning, no matter how, they may find other strategies—such as, attacking other runners. If the runners get into a fist fight, they will all finish late.

Proprietary and secret software is the moral equivalent of runners in a fist fight. Sad to say, the only referee we've got does not seem to object to fights; he just regulates them ("For every ten yards you run, you can fire one shot"). He really ought to break them up, and penalize runners for even trying to fight.

“Won’t everyone stop programming without a monetary incentive?”

Actually, many people will program with absolutely no monetary incentive. Programming has an irresistible fascination for some people, usually the people who are best at it. There is no shortage of professional musicians who keep at it even though they have no hope of making a living that way.

But really this question, though commonly asked, is not appropriate to the situation. Pay for programmers will

not appropriate to the situation. Pay for programmers will not disappear, only become less. So the right question is, will anyone program with a reduced monetary incentive? My experience shows that they will.

For more than ten years, many of the world's best programmers worked at the Artificial Intelligence Lab for far less money than they could have had anywhere else. They got many kinds of nonmonetary rewards: fame and appreciation, for example. And creativity is also fun, a reward in itself.

Then most of them left when offered a chance to do the same interesting work for a lot of money.

What the facts show is that people will program for reasons other than riches; but if given a chance to make a lot of money as well, they will come to expect and demand it. Low-paying organizations do poorly in competition with high-paying ones, but they do not have to do badly if the high-paying ones are banned.

“We need the programmers desperately. If they demand that we stop helping our neighbors, we have to obey.”

have to obey."

You're never so desperate that you have to obey this sort of demand. Remember: millions for defense, but not a cent for tribute!

"Programmers need to make a living somehow."

In the short run, this is true. However, there are plenty of ways that programmers could make a living without selling the right to use a program. This way is customary now because it brings programmers and businessmen the most money, not because it is the only way to make a living. It is easy to find other ways if you want to find them. Here are a number of examples.

A manufacturer introducing a new computer will pay for the porting of operating systems onto the new hardware.

The sale of teaching, handholding and maintenance services could also employ programmers.

People with new ideas could distribute programs as freeware, [10] asking for donations from satisfied users.

freeware, [\[10\]](#) asking for donations from satisfied users, or selling handholding services. I have met people who are already working this way successfully.

Users with related needs can form users' groups, and pay dues. A group would contract with programming companies to write programs that the group's members would like to use.

All sorts of development can be funded with a Software Tax:

Suppose everyone who buys a computer has to pay x percent of the price as a software tax. The government gives this to an agency like the NSF to spend on software development.

But if the computer buyer makes a donation to software development himself, he can take a credit against the tax. He can donate to the project of his own choosing—often, chosen because he hopes to use the results when it is done. He can take a credit for any amount of donation up to the total tax he had to pay.

The total tax rate could be decided by a vote of the payers of the tax, weighted according to the amount they will be taxed on.

The consequences:

- The computer-using community supports software development.
- This community decides what level of support is needed.
- Users who care which projects their share is spent on can choose this for themselves.

In the long run, making programs free is a step toward the postscarcity world, where nobody will have to work very hard just to make a living. People will be free to devote themselves to activities that are fun, such as programming, after spending the necessary ten hours a week on required tasks such as legislation, family counseling, robot repair and asteroid prospecting. There will be no need to be able to make a living from programming.

We have already greatly reduced the amount of

work that the whole society must do for its actual productivity, but only a little of this has translated itself into leisure for workers because much nonproductive activity is required to accompany productive activity. The main causes of this are bureaucracy and isometric struggles against competition. Free software will greatly reduce these drains in the area of software production. We must do this, in order for technical gains in productivity to translate into less work for us.

Endnotes

1 The wording here was careless. The intention was that nobody would have to pay for *permission* to use the GNU system. But the words don't make this clear, and people often interpret them as saying that copies of GNU should always be distributed at little or no charge. That was never the intent; later on, the manifesto mentions the possibility of companies providing the service of distribution for a profit. Subsequently I have learned to distinguish carefully between “free” in the sense of freedom and “free” in the sense of price. Free software is software that users have the freedom to distribute and change. Some users may obtain copies at no charge, while others pay to obtain copies—and if the funds help support improving the software, so much the better. The important thing is that everyone who has a

to make the server. The important thing is that everyone who has a copy has the freedom to cooperate with others in using it.

2 The expression “give away” is another indication that I had not yet clearly separated the issue of price from that of freedom. We now recommend avoiding this expression when talking about free software. See “Words to Avoid (or Use with Care)” ([143](#)) for more explanation.

3 Nowadays, for software tasks to work on, see the High Priority Projects list, at <http://fsf.org/campaigns/priority-projects/>, and the GNU Help Wanted list, the general task list for GNU software packages, at http://savannah.gnu.org/people/?type__id=1. For other ways to help, see <http://gnu.org/help/help/html>.

4 This is another place I failed to distinguish carefully between the two different meanings of “free.” The statement as it stands is not false—you can get copies of GNU software at no charge, from your friends or over the net. But it does suggest the wrong idea.

5 Several such companies now exist.

6 Although it is a charity rather than a company, the Free Software Foundation for 10 years raised most of its funds from its distribution service. You can order things from the FSF to support its work.

7 A group of computer companies pooled funds around 1991 to support maintenance of the GNU C Compiler.

8 I think I was mistaken in saying that proprietary software was the most common basis for making money in software. It seems that actually the most common business model was and is development of custom software. That does not offer the possibility of collecting rents, so the business has to keep doing real work in order to keep getting income. The custom software business would continue to exist, more or less unchanged, in a free software world. Therefore, I no longer expect that most paid programmers would earn less in a free software world.

9 In the 1980s I had not yet realized how confusing it was to speak of “the issue” of “intellectual property.” That term is obviously biased; more subtle is the fact that it lumps together various disparate laws which raise very different issues. Nowadays I urge people to reject the term “intellectual property” entirely, lest it lead others to suppose that those laws form one coherent issue. The way to be clear is to discuss patents, copyrights, and trademarks separately. See “Did You Say ‘Intellectual Property’? It’s a Seductive Mirage” ([137](#)) for further explanation of how this term spreads confusion and bias.

10 Subsequently we learned to distinguish between “free software” and “freeware.” The term “freeware” means software you

are free to redistribute, but usually you are not free to study and change the source code, so most of it is not free software. See “Words to Avoid (or Use with Care)” ([143](#)) for more explanation.

Chapter 5

Why Software Should Not Have Owners

Copyright © 1994, 2009 Richard Stallman

This essay was originally published in *Technos: Quarterly for Education and Technology*, vol. 3, n. 2, pp. 24–26, Summer 1994. This version is published in *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Digital information technology contributes to the world by making it easier to copy and modify information. Computers promise to make this easier for all of us.

Not everyone wants it to be easier. The system of copyright gives software programs “owners,” most of whom aim to withhold software’s potential benefit from the rest of the public. They would like to be the only ones

the rest of the planet. They would like to be the only ones who can copy and modify the software that we use.

The copyright system grew up with printing—a technology for mass-production copying. Copyright fit in well with this technology because it restricted only the mass producers of copies. It did not take freedom away from readers of books. An ordinary reader, who did not own a printing press, could copy books only with pen and ink, and few readers were sued for that.

Digital technology is more flexible than the printing press: when information has digital form, you can easily copy it to share it with others. This very flexibility makes a bad fit with a system like copyright. That's the reason for the increasingly nasty and draconian measures now used to enforce software copyright. Consider these four practices of the Software Publishers Association (SPA):

- Massive propaganda saying it is wrong to disobey the owners to help your friend.
- Solicitation for stool pigeons to inform on their coworkers and colleagues.
- Raids (with police help) on offices and schools, in

which people are told they must prove they are innocent of illegal copying.

- Prosecution (by the US government, at the SPA's request) of people such as MIT's David LaMacchia, not for copying software (he is not accused of copying any), but merely for leaving copying facilities unguarded and failing to censor their use. [\[1\]](#)

All four practices resemble those used in the former Soviet Union, where every copying machine had a guard to prevent forbidden copying, and where individuals had to copy information secretly and pass it from hand to hand as samizdat. There is of course a difference: the motive for information control in the Soviet Union was political; in the US the motive is profit. But it is the actions that affect us, not the motive. Any attempt to block the sharing of information, no matter why, leads to the same methods and the same harshness.

Owners make several kinds of arguments for giving them the power to control how we use information:

Name Calling

Owners use smear words such as “piracy” and “theft,” as well as expert terminology such as “intellectual property” and “damage,” to suggest a certain line of thinking to the public—a simplistic analogy between programs and physical objects.

Our ideas and intuitions about property for material objects are about whether it is right to *take an object away* from someone else. They don’t directly apply to *making a copy* of something. But the owners ask us to apply them anyway.

Exaggeration

Owners say that they suffer “harm” or “economic loss” when users copy programs themselves. But the copying has no direct effect on the owner, and it harms no one. The owner can lose only if the person who made the copy would otherwise have paid for one from the owner.

A little thought shows that most such people would not have bought copies. Yet the owners compute their

“losses” as if each and every one would have bought a copy. That is exaggeration—to put it kindly.

The Law

Owners often describe the current state of the law, and the harsh penalties they can threaten us with. Implicit in this approach is the suggestion that today’s law reflects an unquestionable view of morality—yet at the same time, we are urged to regard these penalties as facts of nature that can’t be blamed on anyone.

This line of persuasion isn’t designed to stand up to critical thinking; it’s intended to reinforce a habitual mental pathway.

It’s elementary that laws don’t decide right and wrong. Every American should know that, in the 1950s, it was against the law in many states for a black person to sit in the front of a bus; but only racists would say sitting there was wrong.

Natural Rights

Authors often claim a special connection with programs they have written, and go on to assert that, as a result, their desires and interests concerning the program simply outweigh those of anyone else—or even those of the whole rest of the world. (Typically companies, not authors, hold the copyrights on software, but we are expected to ignore this discrepancy.)

To those who propose this as an ethical axiom—the author is more important than you—I can only say that I, a notable software author myself, call it bunk.

But people in general are only likely to feel any sympathy with the natural rights claims for two reasons.

One reason is an overstretched analogy with material objects. When I cook spaghetti, I do object if someone else eats it, because then I cannot eat it. His action hurts me exactly as much as it benefits him; only one of us can eat the spaghetti, so the question is, which one? The smallest distinction between us is enough to tip the ethical balance.

But whether you run or change a program I wrote affects you directly and me only indirectly. Whether you give a copy to your friend affects you and your friend much more than it affects me. I shouldn't have the power to tell you not to do these things. No one should.

The second reason is that people have been told that natural rights for authors is the accepted and unquestioned tradition of our society.

As a matter of history, the opposite is true. The idea of natural rights of authors was proposed and decisively rejected when the US Constitution was drawn up. That's why the Constitution only *permits* a system of copyright and does not *require* one; that's why it says that copyright must be temporary. It also states that the purpose of copyright is to promote progress—not to reward authors. Copyright does reward authors somewhat, and publishers more, but that is intended as a means of modifying their behavior.

The real established tradition of our society is that copyright cuts into the natural rights of the public—and that this can only be justified for the public's sake.

Economics

The final argument made for having owners of software is that this leads to production of more software.

Unlike the others, this argument at least takes a legitimate approach to the subject. It is based on a valid goal—satisfying the users of software. And it is empirically clear that people will produce more of something if they are well paid for doing so.

But the economic argument has a flaw: it is based on the assumption that the difference is only a matter of how much money we have to pay. It assumes that *production of software* is what we want, whether the software has owners or not.

People readily accept this assumption because it accords with our experiences with material objects. Consider a sandwich, for instance. You might well be able to get an equivalent sandwich either gratis or for a price. If so, the amount you pay is the only difference. Whether or not you have to buy it, the sandwich has the

whether or not you have to buy it, the sandwich has the same taste, the same nutritional value, and in either case you can only eat it once. Whether you get the sandwich from an owner or not cannot directly affect anything but the amount of money you have afterwards.

This is true for any kind of material object—whether or not it has an owner does not directly affect what it *is*, or what you can do with it if you acquire it.

But if a program has an owner, this very much affects what it is, and what you can do with a copy if you buy one. The difference is not just a matter of money. The system of owners of software encourages software owners to produce something—but not what society really needs. And it causes intangible ethical pollution that affects us all.

What does society need? It needs information that is truly available to its citizens—for example, programs that people can read, fix, adapt, and improve, not just operate. But what software owners typically deliver is a black box that we can't study or change.

Society also needs freedom. When a program has an owner, the users lose freedom to control part of their own lives.

And, above all, society needs to encourage the spirit of voluntary cooperation in its citizens. When software owners tell us that helping our neighbors in a natural way is “piracy,” they pollute our society’s civic spirit.

This is why we say that free software is a matter of freedom, not price.

The economic argument for owners is erroneous, but the economic issue is real. Some people write useful software for the pleasure of writing it or for admiration and love; but if we want more software than those people write, we need to raise funds.

Since the 1980s, free software developers have tried various methods of finding funds, with some success. There’s no need to make anyone rich; a typical income is plenty of incentive to do many jobs that are less satisfying than programming.

For years, until a fellowship made it unnecessary, I made a living from custom enhancements of the free software I had written. Each enhancement was added to the standard released version and thus eventually became available to the general public. Clients paid me so that I would work on the enhancements they wanted, rather than on the features I would otherwise have considered highest priority.

Some free software developers make money by selling support services. In 1994, Cygnus Support, with around 50 employees, estimated that about 15 percent of its staff activity was free software development—a respectable percentage for a software company.

In the early 1990s, companies including Intel, Motorola, Analog Devices Texas Instruments and Analog Devices combined to fund the continued development of the GNU C compiler. Most GCC development is still done by paid developers. The GNU compiler for the Ada language was funded in the 90s by the US Air Force, and continued since then by a company formed specifically for the purpose.

The free software movement is still small, but the example of listener-supported radio in the US shows it's possible to support a large activity without forcing each user to pay.

As a computer user today, you may find yourself using a proprietary program. If your friend asks to make a copy, it would be wrong to refuse. Cooperation is more important than copyright. But underground, closet cooperation does not make for a good society. A person should aspire to live an upright life openly with pride, and this means saying no to proprietary software.

You deserve to be able to cooperate openly and freely with other people who use software. You deserve to be able to learn how the software works, and to teach your students with it. You deserve to be able to hire your favorite programmer to fix it when it breaks.

You deserve free software.

Endnotes

1 The charges were subsequently dismissed.

Chapter 6

Why Software Should Be Free

Copyright © 1991, 1992, 1998, 2000, 2001, 2006, 2007, 2010 Free Software Foundation, Inc.

This version of this essay is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Introduction

The existence of software inevitably raises the question of how decisions about its use should be made. For example, suppose one individual who has a copy of a program meets another who would like a copy. It is possible for them to copy the program, who should decide whether this is done? The individuals involved? Or another party, called the “owner”?

Software developers typically consider these questions on the assumption that the criterion for the answer is to maximize developers’ profits. The political power of business has led to the government adoption of both this criterion and the answer proposed by the developers: that the program has an owner, typically a corporation associated with its development.

I would like to consider the same question using a different criterion: the prosperity and freedom of the

public in general.

This answer cannot be decided by current law—the law should conform to ethics, not the other way around. Nor does current practice decide this question, although it may suggest possible answers. The only way to judge is to see who is helped and who is hurt by recognizing owners of software, why, and how much. In other words, we should perform a cost-benefit analysis on behalf of society as a whole, taking account of individual freedom as well as production of material goods.

In this essay, I will describe the effects of having owners, and show that the results are detrimental. My conclusion is that programmers have the duty to encourage others to share, redistribute, study, and improve the software we write: in other words, to write “free” software. [\[1\]](#)

How Owners Justify Their Power

Those who benefit from the current system where programs are property offer two arguments in support of their claims to own programs: the emotional argument and the economic argument.

The emotional argument goes like this: “I put my sweat, my heart, my soul into this program. It comes from *me*, it’s *mine*!”

This argument does not require serious refutation. The feeling of attachment is one that programmers can cultivate when it suits them; it is not inevitable. Consider, for example, how willingly the same programmers usually sign over all rights to a large corporation for a salary; the

emotional attachment mysteriously vanishes. By contrast, consider the great artists and artisans of medieval times, who didn't even sign their names to their work. To them, the name of the artist was not important. What mattered was that the work was done—and the purpose it would serve. This view prevailed for hundreds of years.

The economic argument goes like this: "I want to get rich"—usually described inaccurately as "making a living"—"and if you don't allow me to get rich by programming, then I won't program. Everyone else is like me, so nobody will ever program. And then you'll be stuck with no programs at all!" This threat is usually veiled as friendly advice from the wise.

I'll explain later why this threat is a bluff. First I want to address an implicit assumption that is more visible in another formulation of the argument.

This formulation starts by comparing the social utility of a proprietary program with that of no program, and then concludes that proprietary software development is, on the whole, beneficial, and should be encouraged. The fallacy here is in comparing only two outcomes—proprietary software versus no software—and assuming there are no other possibilities.

Given a system of software copyright, software development is usually linked with the existence of an owner who controls the software's use. As long as this linkage exists, we are often faced with the choice of proprietary software or none. However, this linkage is not inherent or inevitable; it is a consequence of the specific social/legal policy decision that we are questioning: the decision to have owners. To formulate

the choice as between proprietary software versus no software is begging the question.

The Argument against Having Owners

The question at hand is, "Should development of software be linked with having owners to restrict the use of it?"

In order to decide this, we have to judge the effect on society of each of those two activities *independently*: the effect of developing the software (regardless of its terms of distribution), and the effect of restricting its use (assuming the software has been developed). If one of these activities is helpful and the other is harmful, we would be better off dropping the linkage and doing only the helpful one.

To put it another way, if restricting the distribution of a program already developed is harmful to society overall, then an ethical software developer will reject the option of doing so.

To determine the effect of restricting sharing, we need to compare the value to society of a restricted (i.e., proprietary) program with that of the same program, available to everyone. This means comparing two possible worlds.

This analysis also addresses the simple counterargument sometimes made that "the benefit to the neighbor of giving him or her a copy of a program is cancelled by the harm done to the owner." This counterargument assumes that the harm and the benefit are equal in magnitude. The analysis involves comparing

the equal magnitude. The larger errors comparing these magnitudes, and shows that the benefit is much greater.

To elucidate this argument, let's apply it in another area: road construction.

It would be possible to fund the construction of all roads with tolls. This would entail having toll booths at all street corners. Such a system would provide a great incentive to improve roads. It would also have the virtue of causing the users of any given road to pay for that road. However, a toll booth is an artificial obstruction to smooth driving—artificial, because it is not a consequence of how roads or cars work.

Comparing free roads and toll roads by their usefulness, we find that (all else being equal) roads without toll booths are cheaper to construct, cheaper to run, safer, and more efficient to use. [2] In a poor country, tolls may make the roads unavailable to many citizens. The roads without toll booths thus offer more benefit to society at less cost; they are preferable for society. Therefore, society should choose to fund roads in another way, not by means of toll booths. Use of roads, once built, should be free.

When the advocates of toll booths propose them as *merely* a way of raising funds, they distort the choice that is available. Toll booths do raise funds, but they do something else as well: in effect, they degrade the road. The toll road is not as good as the free road; giving us more or technically superior roads may not be an improvement if this means substituting toll roads for free roads.

Of course, the construction of a free road does cost money, which the public must somehow pay. However, this does not imply the inevitability of toll booths. We who must in either case pay will get more value for our money by buying a free road.

I am not saying that a toll road is worse than no road at all. That would be true if the toll were so great that hardly anyone used the road—but this is an unlikely policy for a toll collector. However, as long as the toll booths cause significant waste and inconvenience, it is better to raise the funds in a less obstructive fashion.

To apply the same argument to software development, I will now show that having “toll booths” for useful software programs costs society dearly: it makes the programs more expensive to construct, more expensive to distribute, and less satisfying and efficient to use. It will follow that program construction should be encouraged in some other way. Then I will go on to explain other methods of encouraging and (to the extent actually necessary) funding software development.

The Harm Done by Obstructing Software

Consider for a moment that a program has been developed, and any necessary payments for its development have been made; now society must choose either to make it proprietary or allow free sharing and use. Assume that the existence of the program and its availability is a desirable thing. [\[3\]](#)

Restrictions on the distribution and modification of the program cannot facilitate its use. They can only

interfere. So the effect can only be negative. But how much? And what kind?

Three different levels of material harm come from such obstruction:

- Fewer people use the program.
- None of the users can adapt or fix the program.
- Other developers cannot learn from the program, or base new work on it.

Each level of material harm has a concomitant form of psychosocial harm. This refers to the effect that people's decisions have on their subsequent feelings, attitudes, and predispositions. These changes in people's ways of thinking will then have a further effect on their relationships with their fellow citizens, and can have material consequences.

The three levels of material harm waste part of the value that the program could contribute, but they cannot reduce it to zero. If they waste nearly all the value of the program, then writing the program harms society by at most the effort that went into writing the program. Arguably a program that is profitable to sell must provide some net direct material benefit.

However, taking account of the concomitant psychosocial harm, there is no limit to the harm that proprietary software development can do.

Obstructing Use of Programs

The first level of harm impedes the simple use of a program. A copy of a program has nearly zero marginal

cost (and you can pay this cost by doing the work yourself), so in a free market, it would have nearly zero price. A license fee is a significant disincentive to use the program. If a widely useful program is proprietary, far fewer people will use it.

It is easy to show that the total contribution of a program to society is reduced by assigning an owner to it. Each potential user of the program, faced with the need to pay to use it, may choose to pay, or may forego use of the program. When a user chooses to pay, this is a zero-sum transfer of wealth between two parties. But each time someone chooses to forego use of the program, this harms that person without benefiting anyone. The sum of negative numbers and zeros must be negative.

But this does not reduce the amount of work it takes to *develop* the program. As a result, the efficiency of the whole process, in delivered user satisfaction per hour of work, is reduced.

This reflects a crucial difference between copies of programs and cars, chairs, or sandwiches. There is no copying machine for material objects outside of science fiction. But programs are easy to copy; anyone can produce as many copies as are wanted, with very little effort. This isn't true for material objects because matter is conserved: each new copy has to be built from raw materials in the same way that the first copy was built.

With material objects, a disincentive to use them makes sense, because fewer objects bought means less raw material and work needed to make them. It's true that there is usually also a startup cost, a development

cost, which is spread over the production run. But as long as the marginal cost of production is significant, adding a share of the development cost does not make a qualitative difference. And it does not require restrictions on the freedom of ordinary users.

However, imposing a price on something that would otherwise be free is a qualitative change. A centrally imposed fee for software distribution becomes a powerful disincentive.

What's more, central production as now practiced is inefficient even as a means of delivering copies of software. This system involves enclosing physical disks or tapes in superfluous packaging, shipping large numbers of them around the world, and storing them for sale. This cost is presented as an expense of doing business; in truth, it is part of the waste caused by having owners.

Damaging Social Cohesion

Suppose that both you and your neighbor would find it useful to run a certain program. In ethical concern for your neighbor, you should feel that proper handling of the situation will enable both of you to use it. A proposal to permit only one of you to use the program, while restraining the other, is divisive; neither you nor your neighbor should find it acceptable.

Signing a typical software license agreement means betraying your neighbor: "I promise to deprive my neighbor of this program so that I can have a copy for myself." People who make such choices feel internal psychological pressure to justify them by downgrading

psychological pressure to justify them, by downgrading the importance of helping one's neighbors—thus public spirit suffers. This is psychosocial harm associated with the material harm of discouraging use of the program.

Many users unconsciously recognize the wrong of refusing to share, so they decide to ignore the licenses and laws, and share programs anyway. But they often feel guilty about doing so. They know that they must break the laws in order to be good neighbors, but they still consider the laws authoritative, and they conclude that being a good neighbor (which they are) is naughty or shameful. That is also a kind of psychosocial harm, but one can escape it by deciding that these licenses and laws have no moral force.

Programmers also suffer psychosocial harm knowing that many users will not be allowed to use their work. This leads to an attitude of cynicism or denial. A programmer may describe enthusiastically the work that he finds technically exciting; then when asked, "Will I be permitted to use it?" his face falls, and he admits the answer is no. To avoid feeling discouraged, he either ignores this fact most of the time or adopts a cynical stance designed to minimize the importance of it.

Since the age of Reagan, the greatest scarcity in the United States is not technical innovation, but rather the willingness to work together for the public good. It makes no sense to encourage the former at the expense of the latter.

Obstructing Custom Adaptation of Programs

The second level of material harm is the inability to adapt programs. The ease of modification of software is one of its great advantages over older technology. But most commercially available software isn't available for modification, even after you buy it. It's available for you to take it or leave it, as a black box—that is all.

A program that you can run consists of a series of numbers whose meaning is obscure. No one, not even a good programmer, can easily change the numbers to make the program do something different.

Programmers normally work with the “source code” for a program, which is written in a programming language such as Fortran or C. It uses names to designate the data being used and the parts of the program, and it represents operations with symbols such as ‘+’ for addition and ‘-’ for subtraction. It is designed to help programmers read and change programs. Here is an example; a program to calculate the distance between two points in a plane:

```
float
distance (p0, p1)
    struct point p0, p1;
{
    float xdist = p1.x - p0.x;
    float ydist = p1.y - p0.y;
    return sqrt (xdist * xdist + ydist * ydist);
}
```

Precisely what that source code means is not the point; the point is that it looks like algebra, and a person who knows this programming language will find it meaningful and clear. By contrast, here is same program in executable form, on the computer I normally used when I wrote this:

1314258944	-232267772	-231844864	1634862
1411907592	-231844736	2159150	1420296208
-234880989	-234879837	-234879966	-232295424
1644167167	-3214848	1090581031	1962942495
572518958	-803143692	1314803317	

Source code is useful (at least potentially) to every user of a program. But most users are not allowed to have copies of the source code. Usually the source code for a proprietary program is kept secret by the owner, lest anybody else learn something from it. Users receive only the files of incomprehensible numbers that the computer will execute. This means that only the program's owner can change the program.

A friend once told me of working as a programmer in a bank for about six months, writing a program similar to something that was commercially available. She believed that if she could have gotten source code for that commercially available program, it could easily have been adapted to their needs. The bank was willing to pay for this, but was not permitted to—the source code was a secret. So she had to do six months of make-work, work that counts in the GNP but was actually waste.

The MIT Artificial Intelligence Lab (AI Lab) received a graphics printer as a gift from Xerox around 1977. It was run by free software to which we added many convenient features. For example, the software would notify a user immediately on completion of a print job. Whenever the printer had trouble, such as a paper jam or running out of paper, the software would immediately notify all users who had print jobs queued. These features facilitated smooth operation.

Later Xerox gave the AI Lab a newer, faster printer, one of the first laser printers. It was driven by proprietary software that ran in a separate dedicated computer, so we couldn't add any of our favorite features. We could arrange to send a notification when a print job was sent to the dedicated computer, but not when the job was actually printed (and the delay was usually considerable). There was no way to find out when the job was actually printed; you could only guess. And no one was informed when there was a paper jam, so the printer often went for an hour without being fixed.

The system programmers at the AI Lab were capable of fixing such problems, probably as capable as the original authors of the program. Xerox was uninterested in fixing them, and chose to prevent us, so we were forced to accept the problems. They were never fixed.

Most good programmers have experienced this frustration. The bank could afford to solve the problem by writing a new program from scratch, but a typical user, no matter how skilled, can only give up.

Giving up causes psychosocial harm—to the spirit of self-reliance. It is demoralizing to live in a house that you cannot rearrange to suit your needs. It leads to resignation and discouragement, which can spread to affect other aspects of one's life. People who feel this way are unhappy and do not do good work.

Imagine what it would be like if recipes were hoarded in the same fashion as software. You might say, "How do I change this recipe to take out the salt?" and the next day you would respond, "How does your family use

the great chef would respond, "How dare you insult my recipe, the child of my brain and my palate, by trying to tamper with it? You don't have the judgment to change my recipe and make it work right!"

"But my doctor says I'm not supposed to eat salt! What can I do? Will you take out the salt for me?"

"I would be glad to do that; my fee is only \$50,000." Since the owner has a monopoly on changes, the fee tends to be large. "However, right now I don't have time. I am busy with a commission to design a new recipe for ship's biscuit for the Navy Department. I might get around to you in about two years."

Obstructing Software Development

The third level of material harm affects software development. Software development used to be an evolutionary process, where a person would take an existing program and rewrite parts of it for one new feature, and then another person would rewrite parts to add another feature; in some cases, this continued over a period of 20 years. Meanwhile, parts of the program would be "cannibalized" to form the beginnings of other programs.

The existence of owners prevents this kind of evolution, making it necessary to start from scratch when developing a program. It also prevents new practitioners from studying existing programs to learn useful techniques or even how large programs can be structured.

Owners also obstruct education. I have met bright students in computer science who have never seen the

source code of a large program. They may be good at writing small programs, but they can't begin to learn the different skills of writing large ones if they can't see how others have done it.

In any intellectual field, one can reach greater heights by standing on the shoulders of others. But that is no longer generally allowed in the software field—you can only stand on the shoulders of the other people *in your own company*.

The associated psychosocial harm affects the spirit of scientific cooperation, which used to be so strong that scientists would cooperate even when their countries were at war. In this spirit, Japanese oceanographers abandoning their lab on an island in the Pacific carefully preserved their work for the invading U.S. Marines, and left a note asking them to take good care of it.

Conflict for profit has destroyed what international conflict spared. Nowadays scientists in many fields don't publish enough in their papers to enable others to replicate the experiment. They publish only enough to let readers marvel at how much they were able to do. This is certainly true in computer science, where the source code for the programs reported on is usually secret.

It Does Not Matter How Sharing Is Restricted

I have been discussing the effects of preventing people from copying, changing, and building on a program. I have not specified how this obstruction is carried out, because that doesn't affect the conclusion. Whether it is done by copyright, or by patent, or by licenses, or

encryption, or ROM cards, or hardware serial numbers, if it *succeeds* in preventing use, it does harm.

Users do consider some of these methods more obnoxious than others. I suggest that the methods most hated are those that accomplish their objective.

Software Should Be Free

I have shown how ownership of a program—the power to restrict changing or copying it—is obstructive. Its negative effects are widespread and important. It follows that society shouldn't have owners for programs.

Another way to understand this is that what society needs is free software, and proprietary software is a poor substitute. Encouraging the substitute is not a rational way to get what we need.

Vaclav Havel has advised us to “Work for something because it is good, not just because it stands a chance to succeed.” A business making proprietary software stands a chance of success in its own narrow terms, but it is not what is good for society.

Why People Will Develop Software

If we eliminate copyright as a means of encouraging people to develop software, at first less software will be developed, but that software will be more useful. It is not clear whether the overall delivered user satisfaction will be less; but if it is, or if we wish to increase it anyway, there are other ways to encourage development, just as there are ways besides toll booths to raise money for

streets. Before I talk about how that can be done, first I want to question how much artificial encouragement is truly necessary.

Programming Is Fun

There are some lines of work that few will enter except for money; road construction, for example. There are other fields of study and art in which there is little chance to become rich, which people enter for their fascination or their perceived value to society. Examples include mathematical logic, classical music, and archaeology; and political organizing among working people. People compete, more sadly than bitterly, for the few funded positions available, none of which is funded very well. They may even pay for the chance to work in the field, if they can afford to.

Such a field can transform itself overnight if it begins to offer the possibility of getting rich. When one worker gets rich, others demand the same opportunity. Soon all may demand large sums of money for doing what they used to do for pleasure. When another couple of years go by, everyone connected with the field will deride the idea that work would be done in the field without large financial returns. They will advise social planners to ensure that these returns are possible, prescribing special privileges, powers, and monopolies as necessary to do so.

This change happened in the field of computer programming in the 1980s. In the 1970s, there were articles on “computer addiction”: users were “onlining” and had hundred-dollar-a-week habits. It was generally understood that people frequently loved programming

enough to break up their marriages. Today, it is generally understood that no one would program except for a high rate of pay. People have forgotten what they knew back then.

When it is true at a given time that most people will work in a certain field only for high pay, it need not remain true. The dynamic of change can run in reverse, if society provides an impetus. If we take away the possibility of great wealth, then after a while, when the people have readjusted their attitudes, they will once again be eager to work in the field for the joy of accomplishment.

The question "How can we pay programmers?" becomes an easier question when we realize that it's not a matter of paying them a fortune. A mere living is easier to raise.

Funding Free Software

Institutions that pay programmers do not have to be software houses. Many other institutions already exist that can do this.

Hardware manufacturers find it essential to support software development even if they cannot control the use of the software. In 1970, much of their software was free because they did not consider restricting it. Today, their increasing willingness to join consortiums shows their realization that owning the software is not what is really important for them.

Universities conduct many programming projects. Today they often sell the results but in the 1970s they

Today they often sell the results, but in the 1970s they did not. Is there any doubt that universities would develop free software if they were not allowed to sell software? These projects could be supported by the same government contracts and grants that now support proprietary software development.

It is common today for university researchers to get grants to develop a system, develop it nearly to the point of completion and call that “finished,” and then start companies where they really finish the project and make it usable. Sometimes they declare the unfinished version “free”; if they are thoroughly corrupt, they instead get an exclusive license from the university. This is not a secret; it is openly admitted by everyone concerned. Yet if the researchers were not exposed to the temptation to do these things, they would still do their research.

Programmers writing free software can make their living by selling services related to the software. I have been hired to port the GNU C compiler to new hardware, and to make user-interface extensions to GNU Emacs. (I offer these improvements to the public once they are done.) I also teach classes for which I am paid.

I am not alone in working this way; there is now a successful, growing corporation which does no other kind of work. Several other companies also provide commercial support for the free software of the GNU system. This is the beginning of the independent software support industry—an industry that could become quite large if free software becomes prevalent. It provides users with an option generally unavailable for proprietary software, except to the very wealthy.

Institutions such as the Free Software Foundation can also fund programmers. Most of the Foundation's funds come from users buying tapes through the mail. The software on the tapes is free, which means that every user has the freedom to copy it and change it, but many nonetheless pay to get copies. (Recall that "free software" refers to freedom, not to price.) Some users who already have a copy order tapes as a way of making a contribution they feel we deserve. The Foundation also receives sizable donations from computer manufacturers.

The Free Software Foundation is a charity, and its income is spent on hiring as many programmers as possible. If it had been set up as a business, distributing the same free software to the public for the same fee, it would now provide a very good living for its founder.

Because the Foundation is a charity, programmers often work for the Foundation for half of what they could make elsewhere. They do this because we are free of bureaucracy, and because they feel satisfaction in knowing that their work will not be obstructed from use. Most of all, they do it because programming is fun. In addition, volunteers have written many useful programs for us. (Even technical writers have begun to volunteer.)

This confirms that programming is among the most fascinating of all fields, along with music and art. We don't have to fear that no one will want to program.

What Do Users Owe to Developers?

There is a good reason for users of software to feel a moral obligation to contribute to its support. Developers

of free software are contributing to the users' activities, and it is both fair and in the long-term interest of the users to give them funds to continue.

However, this does not apply to proprietary software developers, since obstructionism deserves a punishment rather than a reward.

We thus have a paradox: the developer of useful software is entitled to the support of the users, but any attempt to turn this moral obligation into a requirement destroys the basis for the obligation. A developer can either deserve a reward or demand it, but not both.

I believe that an ethical developer faced with this paradox must act so as to deserve the reward, but should also entreat the users for voluntary donations. Eventually the users will learn to support developers without coercion, just as they have learned to support public radio and television stations.

What Is Software Productivity?

If software were free, there would still be programmers, but perhaps fewer of them. Would this be bad for society?

Not necessarily. Today the advanced nations have fewer farmers than in 1900, but we do not think this is bad for society, because the few deliver more food to the consumers than the many used to do. We call this improved productivity. Free software would require far fewer programmers to satisfy the demand, because of increased software productivity at all levels:

- Wider use of each program that is developed.
- The ability to adapt existing programs for customization instead of starting from scratch.
- Better education of programmers.
- The elimination of duplicate development effort.

Those who object to cooperation claiming it would result in the employment of fewer programmers are actually objecting to increased productivity. Yet these people usually accept the widely held belief that the software industry needs increased productivity. How is this?

“Software productivity” can mean two different things: the overall productivity of all software development, or the productivity of individual projects. Overall productivity is what society would like to improve, and the most straightforward way to do this is to eliminate the artificial obstacles to cooperation which reduce it. But researchers who study the field of “software productivity” focus only on the second, limited, sense of the term, where improvement requires difficult technological advances.

Is Competition Inevitable?

Is it inevitable that people will try to compete, to surpass their rivals in society? Perhaps it is. But competition itself is not harmful; the harmful thing is *combat*.

There are many ways to compete. Competition can consist of trying to achieve ever more, to outdo what others have done. For example, in the old days, there was competition among programming wizards—competition for who could make the computer do the

competition for who could make the computer do the most amazing thing, or for who could make the shortest or fastest program for a given task. This kind of competition can benefit everyone, *as long as* the spirit of good sportsmanship is maintained.

Constructive competition is enough competition to motivate people to great efforts. A number of people are competing to be the first to have visited all the countries on Earth; some even spend fortunes trying to do this. But they do not bribe ship captains to strand their rivals on desert islands. They are content to let the best person win.

Competition becomes combat when the competitors begin trying to impede each other instead of advancing themselves—when “Let the best person win” gives way to “Let me win, best or not.” Proprietary software is harmful, not because it is a form of competition, but because it is a form of combat among the citizens of our society.

Competition in business is not necessarily combat. For example, when two grocery stores compete, their entire effort is to improve their own operations, not to sabotage the rival. But this does not demonstrate a special commitment to business ethics; rather, there is little scope for combat in this line of business short of physical violence. Not all areas of business share this characteristic. Withholding information that could help everyone advance is a form of combat.

Business ideology does not prepare people to resist the temptation to combat the competition. Some forms of combat have been banned with antitrust laws, truth in advertising laws, and so on, but rather than generalizing

advertising laws, and so on, but rather than generalizing this to a principled rejection of combat in general, executives invent other forms of combat which are not specifically prohibited. Society's resources are squandered on the economic equivalent of factional civil war.

“Why Don't You Move to Russia?”

In the United States, any advocate of other than the most extreme form of laissez-faire selfishness has often heard this accusation. For example, it is leveled against the supporters of a national health care system, such as is found in all the other industrialized nations of the free world. It is leveled against the advocates of public support for the arts, also universal in advanced nations. The idea that citizens have any obligation to the public good is identified in America with Communism. But how similar are these ideas?

Communism as was practiced in the Soviet Union was a system of central control where all activity was regimented, supposedly for the common good, but actually for the sake of the members of the Communist party. And where copying equipment was closely guarded to prevent illegal copying.

The American system of software copyright exercises central control over distribution of a program, and guards copying equipment with automatic copying-protection schemes to prevent illegal copying.

By contrast, I am working to build a system where people are free to decide their own actions; in particular, free to help their neighbors, and free to alter and improve

the tools which they use in their daily lives. A system based on voluntary cooperation and on decentralization.

Thus, if we are to judge views by their resemblance to Russian Communism, it is the software owners who are the Communists.

The Question of Premises

I make the assumption in this paper that a user of software is no less important than an author, or even an author's employer. In other words, their interests and needs have equal weight, when we decide which course of action is best.

This premise is not universally accepted. Many maintain that an author's employer is fundamentally more important than anyone else. They say, for example, that the purpose of having owners of software is to give the author's employer the advantage he deserves—regardless of how this may affect the public.

It is no use trying to prove or disprove these premises. Proof requires shared premises. So most of what I have to say is addressed only to those who share the premises I use, or at least are interested in what their consequences are. For those who believe that the owners are more important than everyone else, this paper is simply irrelevant.

But why would a large number of Americans accept a premise that elevates certain people in importance above everyone else? Partly because of the belief that this premise is part of the legal traditions of American society. Some people feel that doubting the premise

means challenging the basis of society.

It is important for these people to know that this premise is not part of our legal tradition. It never has been.

Thus, the Constitution says that the purpose of copyright is to “promote the Progress of Science and the useful Arts.” The Supreme Court has elaborated on this, stating in *Fox Film v. Doyal* [4] that “The sole interest of the United States and the primary object in conferring the [copyright] monopoly lie in the general benefits derived by the public from the labors of authors.”

We are not required to agree with the Constitution or the Supreme Court. (At one time, they both condoned slavery.) So their positions do not disprove the owner supremacy premise. But I hope that the awareness that this is a radical right-wing assumption rather than a traditionally recognized one will weaken its appeal.

Conclusion

We like to think that our society encourages helping your neighbor; but each time we reward someone for obstructionism, or admire them for the wealth they have gained in this way, we are sending the opposite message.

Software hoarding is one form of our general willingness to disregard the welfare of society for personal gain. We can trace this disregard from Ronald Reagan to Dick Cheney, from Exxon to Enron, from failing banks to failing schools. We can measure it with the size of the homeless population and the prison population. The antisocial spirit feeds on itself because

population. The universal spirit leads on itself, because the more we see that other people will not help us, the more it seems futile to help them. Thus society decays into a jungle.

If we don't want to live in a jungle, we must change our attitudes. We must start sending the message that a good citizen is one who cooperates when appropriate, not one who is successful at taking from others. I hope that the free software movement will contribute to this: at least in one area, we will replace the jungle with a more efficient system which encourages and runs on voluntary cooperation.

Endnotes

1 The word “free” in “free software” refers to freedom, not to price; the price paid for a copy of a free program may be zero, or small, or (rarely) quite large.

2 The issues of pollution and traffic congestion do not alter this conclusion. If we wish to make driving more expensive to discourage driving in general, it is disadvantageous to do this using toll booths, which contribute to both pollution and congestion. A tax on gasoline is much better. Likewise, a desire to enhance safety by limiting maximum speed is not relevant; a free-access road enhances the average speed by avoiding stops and delays, for any given speed limit.

3 One might regard a particular computer program as a harmful thing that should not be available at all, like the Lotus Marketplace database of personal information, which was withdrawn from sale due to public disapproval. Most of what I say does not apply to this case, but it makes little sense to argue for having an owner on the grounds that the owner will make the program less available. The owner will not make it *completely* unavailable, as one would wish in the case of a program whose use is considered destructive.

4 *Fox Film Corp. v. Doyal*, 286 US 123, 1932.

Chapter 7

Why Schools Should Exclusively Use Free Software

Copyright © 2003, 2009 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2003.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

There are general reasons why all computer users should insist on free software: it gives users the freedom to control their own computers—with proprietary software, the computer does what the software owner wants it to do, not what the user wants it to do. Free software also gives users the freedom to cooperate with each other, to lead an upright life. These reasons apply to schools as they do to everyone.

The purpose of this article is to state additional reasons that apply specifically to education.

First, free software can save schools money. Free software gives schools, like other users, the freedom to copy and redistribute the software, so the school system can make copies for all the computers they have. In poor countries, this can help close the digital divide.

This obvious reason, while important in practical terms, is rather shallow. And proprietary software developers can eliminate this reason by donating copies to the schools. (Warning: a school that accepts such an offer may have to pay for upgrades later.) So let's look at the deeper reasons.

Schools have a social mission: to teach students to be citizens of a strong, capable, independent, cooperating and free society. They should promote the use of free software just as they promote recycling. If schools teach students free software, then the students will tend to use free software after they graduate. This will help society as a whole escape from being dominated (and gouged) by megacorporations.

What schools should refuse to do is teach dependence. Those corporations offer free samples to schools for the same reason tobacco companies distribute free cigarettes to minors: to get children addicted. [\[1\]](#) They will not give discounts to these students once they've grown up and graduated.

Free software permits students to learn how software works. Some students, on reaching their teens, want to learn everything there is to know about their computer and its software. They are intensely curious to read the source code of the programs that they use every day. To learn to write good code, students need to read lots of code and write lots of code. They need to read and understand real programs that people really use. Only free software permits this

Proprietary software rejects their thirst for knowledge: it says, "The knowledge you want is a secret—learning is forbidden!" Free software encourages everyone to learn. The free software community rejects the "priesthood of technology," which keeps the general public in ignorance of how technology works; we encourage students of any age and situation to read the source code and learn as much as they want to know. Schools that use free software will enable gifted programming students to advance.

The deepest reason for using free software in schools is for moral education. We expect schools to teach students basic facts and useful skills, but that is not their whole job. The most fundamental job of schools is to teach good citizenship, which includes the habit of helping others. In the area of computing, this means teaching people to share software. Schools, starting from nursery school, should tell their pupils, "If you bring software to school, you must share it with the other students. And you must show the source code to the class, in case someone wants to learn."

Of course, the school must practice what it preaches: all the software installed by the school should be available for students to copy, take home, and redistribute further.

Teaching the students to use free software, and to participate in the free software community, is a hands-on civics lesson. It also teaches students the role model of

public service rather than that of tycoons. All levels of school should use free software.

Endnotes

1 RJ Reynolds Tobacco Company was fined \$15m in 2002 for handing out free samples of cigarettes at events attended by children. See

http://www.bbc.co.uk/worldservice/sci_tech/features/health/tobaccotrial/usa.htm.

Chapter 8

Releasing Free Software If You Work at a University

Copyright © 2002 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2002.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

In the free software movement, we believe computer users should have the freedom to change and redistribute the software that they use. The “free” in “free software” refers to freedom: it means users have the freedom to run, modify and redistribute the software. Free software contributes to human knowledge, while nonfree software does not. Universities should therefore encourage free software for the sake of advancing human knowledge,

just as they should encourage scientists and other scholars to publish their work.

Alas, many university administrators have a grasping attitude towards software (and towards science); they see programs as opportunities for income, not as opportunities to contribute to human knowledge. Free software developers have been coping with this tendency for almost 20 years.

When I started developing the GNU operating system, in 1984, my first step was to quit my job at MIT. I did this specifically so that the MIT licensing office would be unable to interfere with releasing GNU as free software. I had planned an approach for licensing the programs in GNU that would ensure that all modified versions must be free software as well—an approach that developed into the GNU General Public License (GNU GPL)—and I did not want to have to beg the MIT administration to let me use it.

Over the years, university affiliates have often come to the Free Software Foundation for advice on how to cope with administrators who see software only as

something to sell. One good method, applicable even for specifically funded projects, is to base your work on an existing program that was released under the GNU GPL. Then you can tell the administrators, “We’re not allowed to release the modified version except under the GNU GPL—any other way would be copyright infringement.” After the dollar signs fade from their eyes, they will usually consent to releasing it as free software.

You can also ask your funding sponsor for help. When a group at NYU developed the GNU Ada Compiler, with funding from the US Air Force, the contract explicitly called for donating the resulting code to the Free Software Foundation. Work out the arrangement with the sponsor first, then politely show the university administration that it is not open to renegotiation. They would rather have a contract to develop free software than no contract at all, so they will most likely go along.

Whatever you do, raise the issue early—well before the program is half finished. At this point, the university still needs you, so you can play hardball: tell the

administration you will finish the program, make it usable, if they agree in writing to make it free software (and agree to your choice of free software license). Otherwise you will work on it only enough to write a paper about it, and never make a version good enough to release. When the administrators know their choice is to have a free software package that brings credit to the university or nothing at all, they will usually choose the former.

Not all universities have grasping policies. The University of Texas has a policy that makes it easy to release software developed there as free software under the GNU General Public License. Univates, in Brazil, and the International Institute of Information Technology in Hyderabad, India, both have policies in favor of releasing software under the GPL. By developing faculty support first, you may be able to institute such a policy at your university. Present the issue as one of principle: does the university have a mission to advance human knowledge, or is its sole purpose to perpetuate itself?

Whatever approach you use, it helps to approach the issue with determination and based on an ethical perspective, as we do in the free software movement. To

proprietary, as we do in the case of software. To treat the public ethically, the software should be free—as in freedom—for the whole public.

Many developers of free software profess narrowly practical reasons for doing so: they advocate allowing others to share and change software as an expedient for making software powerful and reliable. If those values motivate you to develop free software, well and good, and thank you for your contribution. But those values do not give you a good footing to stand firm when university administrators pressure or tempt you to make the program nonfree.

For instance, they may argue that “We could make it even more powerful and reliable with all the money we can get.” This claim may or may not come true in the end, but it is hard to disprove in advance. They may suggest a license to offer copies “free of charge, for academic use only,” which would tell the general public they don’t deserve freedom, and argue that this will obtain the cooperation of academia, which is all (they say) you need.

If you start from values of convenience alone, it is hard to make a good case for rejecting these dead-end proposals, but you can do it easily if you base your stand on ethical and political values. What good is it to make a program powerful and reliable at the expense of users' freedom? Shouldn't freedom apply outside academia as well as within it? The answers are obvious if freedom and community are among your goals. Free software respects the users' freedom, while nonfree software negates it.

Nothing strengthens your resolve like knowing that the community's freedom depends, in one instance, on you.

Chapter 9

Why Free Software Needs Free Documentation

Copyright © 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2009 Free Software Foundation, Inc.

This essay was originally published on <http://gnu.org>, in 1996. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

The biggest deficiency in free operating systems is not in the software—it is the lack of good free manuals that we can include in these systems. Many of our most important programs do not come with full manuals. Documentation is an essential part of any software package; when an important free software package does not come with a free manual, that is a major gap. We

have many such gaps today.

Once upon a time, many years ago, I thought I would learn Perl. I got a copy of a free manual, but I found it hard to read. When I asked Perl users about alternatives, they told me that there were better introductory manuals—but those were not free.

Why was this? The authors of the good manuals had written them for O'Reilly Associates, which published them with restrictive terms—no copying, no modification, source files not available—which exclude them from the free software community.

That wasn't the first time this sort of thing has happened, and (to our community's great loss) it was far from the last. Proprietary manual publishers have enticed a great many authors to restrict their manuals since then. Many times I have heard a GNU user eagerly tell me about a manual that he is writing, with which he expects to help the GNU Project—and then had my hopes dashed, as he proceeded to explain that he had signed a contract with a publisher that would restrict it so that we cannot use it.

Given that writing good English is a rare skill among programmers, we can ill afford to lose manuals this way.

Free documentation, like free software, is a matter of freedom, not price. The problem with these manuals was not that O'Reilly Associates charged a price for printed copies—that in itself is fine. (The Free Software Foundation sells printed copies of free GNU manuals, too.) But GNU manuals are available in source code form, while these manuals are available only on paper. GNU manuals come with permission to copy and modify; the Perl manuals do not. These restrictions are the problems.

The criterion for a free manual is pretty much the same as for free software: it is a matter of giving all users certain freedoms. Redistribution (including commercial redistribution) must be permitted, so that the manual can accompany every copy of the program, on line or on paper. Permission for modification is crucial too.

As a general rule, I don't believe that it is essential for people to have permission to modify all sorts of

for people to have permission to modify all sorts of articles and books. The issues for writings are not necessarily the same as those for software. For example, I don't think you or I are obliged to give permission to modify articles like this one, which describe our actions and our views.

But there is a particular reason why the freedom to modify is crucial for documentation for free software. When people exercise their right to modify the software, and add or change its features, if they are conscientious they will change the manual too—so they can provide accurate and usable documentation with the modified program. A manual which forbids programmers from being conscientious and finishing the job, or more precisely requires them to write a new manual from scratch if they change the program, does not fill our community's needs.

While a blanket prohibition on modification is unacceptable, some kinds of limits on the method of modification pose no problem. For example, requirements to preserve the original author's copyright notice, the distribution terms, or the list of authors, are

OK. It is also no problem to require modified versions to include notice that they were modified, even to have entire sections that may not be deleted or changed, as long as these sections deal with nontechnical topics. (Some GNU manuals have them.)

These kinds of restrictions are not a problem because, as a practical matter, they don't stop the conscientious programmer from adapting the manual to fit the modified program. In other words, they don't block the free software community from making full use of the manual.

However, it must be possible to modify all the *technical* content of the manual, and then distribute the result through all the usual media, through all the usual channels; otherwise, the restrictions do block the community, the manual is not free, and so we need another manual.

Unfortunately, it is often hard to find someone to write another manual when a proprietary manual exists. The obstacle is that many users think that a proprietary manual is good enough—so they don't see the need to

manual is good enough - so they don't see the need to write a free manual. They do not see that the free operating system has a gap that needs filling.

Why do users think that proprietary manuals are good enough? Some have not considered the issue. I hope this article will do something to change that.

Other users consider proprietary manuals acceptable for the same reason so many people consider proprietary software acceptable: they judge in purely practical terms, not using freedom as a criterion. These people are entitled to their opinions, but since those opinions spring from values which do not include freedom, they are no guide for those of us who do value freedom.

Please spread the word about this issue. We continue to lose manuals to proprietary publishing. If we spread the word that proprietary manuals are not sufficient, perhaps the next person who wants to help GNU by writing documentation will realize, before it is too late, that he must above all make it free.

We can also encourage commercial publishers to sell free manuals instead of proprietary ones.

free, copylefted manuals instead of proprietary ones. One way you can help this is to check the distribution terms of a manual before you buy it, and prefer copylefted manuals to noncopylefted ones.

Note: We maintain a page that lists free books available from other publishers.

Chapter 10

Selling Free Software

Some views on the ideas of selling exceptions to free software licenses, such as the GNU GPL, are also available, at

<http://gnu.org/philosophy/selling-exceptions/html>.

Copyright © 1996, 1997, 1998, 2001, 2007 Free Software Foundation, Inc.

This essay was originally published on <http://gnu.org>, in 1996.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Many people believe that the spirit of the GNU Project is that you should not charge money for distributing copies of software, or that you should charge as little as possible—just enough to cover the cost. This is a misunderstanding.

Actually, we encourage people who redistribute free

software to charge as much as they wish or can. If this seems surprising to you, please read on.

The word “free” has two legitimate general meanings; it can refer either to freedom or to price. When we speak of “free software,” we’re talking about freedom, not price. (Think of “free speech,” not “free beer.”) Specifically, it means that a user is free to run the program, change the program, and redistribute the program with or without changes.

Free programs are sometimes distributed gratis, and sometimes for a substantial price. Often the same program is available in both ways from different places. The program is free regardless of the price, because users have freedom in using it.

Nonfree programs are usually sold for a high price, but sometimes a store will give you a copy at no charge. That doesn’t make it free software, though. Price or no price, the program is nonfree because users don’t have freedom.

Since free software is not a matter of price, a low price doesn’t make the software free, or even closer to free. So if you are redistributing copies of free software.

you might as well charge a substantial fee and *make some money*. Redistributing free software is a good and legitimate activity; if you do it, you might as well make a profit from it.

Free software is a community project, and everyone who depends on it ought to look for ways to contribute to building the community. For a distributor, the way to do this is to give a part of the profit to free software development projects or to the Free Software Foundation. This way you can advance the world of free software.

Distributing free software is an opportunity to raise funds for development. Don't waste it!

In order to contribute funds, you need to have some extra. If you charge too low a fee, you won't have anything to spare to support development.

Will a Higher Distribution Price Hurt Some Users?

People sometimes worry that a high distribution fee will put free software out of range for users who don't have a lot of money. With proprietary software, a high price

lot of money. With proprietary software, a high price does exactly that—but free software is different.

The difference is that free software naturally tends to spread around, and there are many ways to get it.

Software hoarders try their damndest to stop you from running a proprietary program without paying the standard price. If this price is high, that does make it hard for some users to use the program.

With free software, users don't *have* to pay the distribution fee in order to use the software. They can copy the program from a friend who has a copy, or with the help of a friend who has network access. Or several users can join together, split the price of one CD-ROM, then each in turn can install the software. A high CD-ROM price is not a major obstacle when the software is free.

Will a Higher Distribution Price Discourage Use of Free Software?

Another common concern is for the popularity of free software. People think that a high price for distribution would reduce the number of users, or that a low price is

likely to encourage users.

This is true for proprietary software—but free software is different. With so many ways to get copies, the price of distribution service has less effect on popularity.

In the long run, how many people use free software is determined mainly by *how much free software can do*, and how easy it is to use. Many users do not make freedom their priority; they may continue to use proprietary software if free software can't do all the jobs they want done. Thus, if we want to increase the number of users in the long run, we should above all *develop more free software*.

The most direct way to do this is by writing needed free software or manuals yourself. But if you do distribution rather than writing, the best way you can help is by raising funds for others to write them.

The Term “Selling Software” Can Be Confusing Too

Strictly speaking, “selling” means trading goods for money. Selling a copy of a free program is legitimate, and

we encourage it.

However, when people think of “selling software,” they usually imagine doing it the way most companies do it: making the software proprietary rather than free.

So unless you’re going to draw distinctions carefully, the way this article does, we suggest it is better to avoid using the term “selling software” and choose some other wording instead. For example, you could say “distributing free software for a fee”—that is unambiguous.

High or Low Fees, and the GNU GPL

Except for one special situation, the GNU General Public License (GNU GPL) has no requirements about how much you can charge for distributing a copy of free software. You can charge nothing, a penny, a dollar, or a billion dollars. It’s up to you, and the marketplace, so don’t complain to us if nobody wants to pay a billion dollars for a copy.

The one exception is in the case where binaries are distributed without the corresponding complete source code. Those who do this are required by the GNU GPL

code. Those who do this are required by the GNU GPL to provide source code on subsequent request. Without a limit on the fee for the source code, they would be able to set a fee too large for anyone to pay—such as a billion dollars—and thus pretend to release source code while in truth concealing it. So in this case we have to limit the fee for source in order to ensure the user's freedom. In ordinary situations, however, there is no such justification for limiting distribution fees, so we do not limit them.

Sometimes companies whose activities cross the line stated in the GNU GPL plead for permission, saying that they “won’t charge money for the GNU software” or such like. That won’t get them anywhere with us. Free software is about freedom, and enforcing the GPL is defending freedom. When we defend users’ freedom, we are not distracted by side issues such as how much of a distribution fee is charged. Freedom is the issue, the whole issue, and the only issue.

Chapter 11

The Free Software Song

This song is in a rhythm of 7/8; those unaccustomed to odd rhythms often take the unevenness to be a mistake. The meter can be analyzed into three subgroups as slow-quick-quick or 3-2-2. Such meters in Bulgarian music can often be stretched, and some musicians analyze this song as 3-2-3 instead; however, the last “3” is not as long as the first. Yves Moreau, who collected and taught the dance, endorses the rhythm of 7.

Copyright © 2010 Richard Stallman

Richard Stallman wrote the lyrics above in 1991. This version of the score is published in *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

The lyrics of “The Free Software Song” are sung to the melody of the Bulgarian folk song “Sadi moma bela loza.” To listen to a recording of the piece accompanied by Bulgarian instruments played in traditional style, please visit <http://gnu.org/music/FreeSWSong.ogg>.



Join us now and share the soft-ware; you'll be ____ free, ha-ckers,
Hoar-ders can get piles of mo - ney; that is ____ true, ha-ckers,
When we have e - nough free soft-ware at our ____ call, ha-ckers,
Join us now and share the soft-ware; you'll be ____ free, ha-ckers,



you'll be free. Join us now and share the soft - ware;
 that is true. But they can - not help their neigh-bors;
 at our call, we'll kick out those dir - ty li - cen-ses
 you'll be free. Join us now and share the soft - ware;

9

you'll be free, ha - ckers, you'll be free.
 that's not good, ha - ckers, that's not good.
 e - ver more, ha - ckers, e - ver more.
 you'll be free, ha - ckers, you'll be free.

Part II

What's in a Name?

Chapter 12

What's in a Name?

To learn more about this issue, you can read our GNU/Linux FAQ, at <http://gnu.org/gnu/gnu-linux-faq.html>, our page on Linux and the GNU Project, at <http://gnu.org/gnu/linux-and-gnu.html>, which gives a history of the GNU/Linux system as it relates to this issue of naming, and the article “GNU Users Who Have Never Heard of GNU,” at <http://gnu.org/gnu/gnu-users-never-heard-of-gnu.html>.

Copyright © 2000, 2006, 2007 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2000.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Names convey meanings; our choice of names determines the meaning of what we say. An inappropriate name gives people the wrong idea. A rose by any other name would smell as sweet—but if you call

it a pen, people will be rather disappointed when they try to write with it. And if you call pens “roses,” people may not realize what they are good for. If you call our operating system Linux, that conveys a mistaken idea of the system’s origin, history, and purpose. If you call it GNU/Linux, that conveys (though not in detail) an accurate idea.

Does this really matter for our community? Is it important whether people know the system’s origin, history, and purpose? Yes—because people who forget history are often condemned to repeat it. The Free World that has developed around GNU/Linux is not guaranteed to survive; the problems that led us to develop GNU are not completely eradicated, and they threaten to come back.

When I explain why it’s appropriate to call the operating system GNU/Linux rather than Linux, people sometimes respond this way:

Granted that the GNU Project deserves credit for this work, is it really worth a fuss when people don’t give credit? Isn’t the important thing that the job was done, not who did it? You ought to relax, take pride in the job well done, and not worry about the credit.

This would be wise advice, if only the situation were like that—if the job were done and it were time to relax. If only that were true! But challenges abound, and this is no time to take the future for granted. Our community's strength rests on commitment to freedom and cooperation. Using the name GNU/Linux is a way for people to remind themselves and inform others of these goals.

It is possible to write good free software without thinking of GNU; much good work has been done in the name of Linux also. But the term “Linux” has been associated ever since it was first coined with a philosophy that does not make a commitment to the freedom to cooperate. As the name is increasingly used by business, we will have even more trouble making it connect with community spirit.

A great challenge to the future of free software comes from the tendency of the “Linux” distribution companies to add nonfree software to GNU/Linux in the name of convenience and power. All the major commercial distribution developers do this; none limits itself to free software. Most of them do not clearly identify the nonfree packages in their distributions. Many

even develop nonfree software and add it to the system. Some outrageously advertise “Linux” systems that are “licensed per seat,” which give the user as much freedom as Microsoft Windows.

People try to justify adding nonfree software in the name of the “popularity of Linux”—in effect, valuing popularity above freedom. Sometimes this is openly admitted. For instance, *Wired* magazine said that Robert McMillan, editor of *Linux Magazine*, “feels that the move toward open source software should be fueled by technical, rather than political, decisions.” [\[1\]](#) And Caldera’s CEO openly urged users to drop the goal of freedom and work instead for the “popularity of Linux.”

Adding nonfree software to the GNU/Linux system may increase the popularity, if by popularity we mean the number of people using some of GNU/Linux in combination with nonfree software. But at the same time, it implicitly encourages the community to accept nonfree software as a good thing, and forget the goal of freedom. It is not good to drive faster if you can’t stay on the road.

When the nonfree “add-on” is a library or programming tool, it can become a trap for free software developers. When they write free software that depends

on the nonfree package, their software cannot be part of a completely free system. Motif and Qt trapped large amounts of free software in this way in the past, creating problems whose solutions took years. Motif remained somewhat of a problem until it became obsolete and was no longer used. Later, Sun's nonfree Java implementation had a similar effect: the Java Trap, fortunately now mostly corrected.

If our community keeps moving in this direction, it could redirect the future of GNU/Linux into a mosaic of free and nonfree components. Five years from now, we will surely still have plenty of free software; but if we are not careful, it will hardly be usable without the nonfree software that users expect to find with it. If this happens, our campaign for freedom will have failed.

If releasing free alternatives were simply a matter of programming, solving future problems might become easier as our community's development resources increase. But we face obstacles that threaten to make this harder: laws that prohibit free software. As software patents mount up, and as laws like the Digital Millennium Copyright Act are used to prohibit the development of free software for important jobs such as viewing a DVD or listening to a RealAudio stream, we will find ourselves

or listening to a RealAudio stream, we will find ourselves with no clear way to fight the patented and secret data formats except to *reject the nonfree programs that use them*.

Meeting these challenges will require many different kinds of effort. But what we need above all, to confront any kind of challenge, is to remember the goal of freedom to cooperate. We can't expect a mere desire for powerful, reliable software to motivate people to make great efforts. We need the kind of determination that people have when they fight for their freedom and their community—determination to keep on for years and not give up.

In our community, this goal and this determination emanate mainly from the GNU Project. We're the ones who talk about freedom and community as something to stand firm for; the organizations that speak of "Linux" normally don't say this. The magazines about "Linux" are typically full of ads for nonfree software; the companies that package "Linux" add nonfree software to the system; other companies "support Linux" by developing nonfree applications to run on GNU/Linux; the user groups for "Linux" typically invite salesmen to present those applications. The main place people in our

community are likely to come across the idea of freedom and determination is in the GNU Project.

But when people come across it, will they feel it relates to them?

People who know they are using a system that came out of the GNU Project can see a direct relationship between themselves and GNU. They won't automatically agree with our philosophy, but at least they will see a reason to think seriously about it. In contrast, people who consider themselves "Linux users," and believe that the GNU Project "developed tools which proved to be useful in Linux," typically perceive only an indirect relationship between GNU and themselves. They may just ignore the GNU philosophy when they come across it.

The GNU Project is idealistic, and anyone encouraging idealism today faces a great obstacle: the prevailing ideology encourages people to dismiss idealism as "impractical." Our idealism has been extremely practical: it is the reason we have a free GNU/Linux operating system. People who love this system ought to know that it is our idealism made real.

If “the job” really were done, if there were nothing at stake except credit, perhaps it would be wiser to let the matter drop. But we are not in that position. To inspire people to do the work that needs to be done, we need to be recognized for what we have already done. Please help us, by calling the operating system GNU/Linux.

Endnotes

¹ Michelle Finley, “French Pols Say, ‘Open It Up,’”
24 April 2000,
<http://wired.com/politics/law/news/2000/04/35862>.

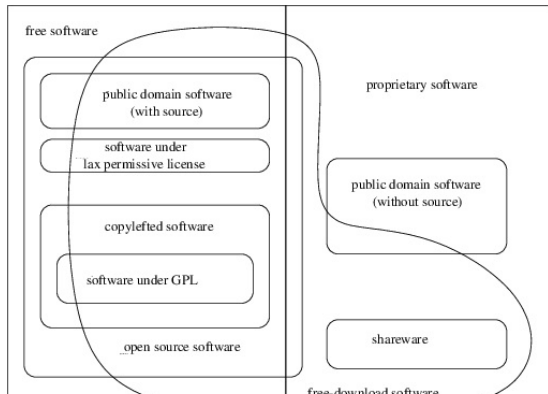
Chapter 13

Categories of Free and Nonfree Software

Copyright © 1996, 1997, 1998, 2001, 2006, 2007, 2009, 2010 Free Software Foundation, Inc.

This list was originally published on <http://gnu.org>, in 1996. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.



This diagram, originally by Chao-Kuei and updated by several others since, explains the different categories of software. It's available at <http://gnu.org/philosophy/categories.html> as a Scalable Vector Graphic and as an XFig document, under the terms of any of the GNU GPL v2 or later, the GNU FDL v1.2 or later, or the Creative Commons Attribution-Share Alike v2.0 or later. To view a copy of the Creative Commons license, visit <http://creativecommons.org/licenses/by-sa/2.0>, or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

Free Software

Free software is software that comes with permission for anyone to use, copy, and/or distribute, either verbatim or with modifications, either gratis or for a fee. In particular, this means that source code must be available. “If it’s not source, it’s not software.” This is a simplified description; see also the full definition (9).

If a program is free, then it can potentially be included in a free operating system such as GNU, or free versions of the GNU/Linux system.

There are many different ways to make a program free—many questions of detail, which could be decided in more than one way and still make the program free. Some of the possible variations are described below. For information on specific free software licenses, see the license list page, at <http://gnu.org/licenses/license-list.html>.

Free software is a matter of freedom, not price. But nonproprietary software companies typically use the term

properly, software companies typically use the term “free software” to refer to price. Sometimes they mean that you can obtain a binary copy at no charge; sometimes they mean that a copy is bundled with a computer that you are buying, and the price includes both. Either way, it has nothing to do with what we mean by free software in the GNU Project.

Because of this potential confusion, when a software company says its product is free software, always check the actual distribution terms to see whether users really have all the freedoms that free software implies. Sometimes it really is free software; sometimes it isn't.

Many languages have two separate words for “free” as in freedom and “free” as in zero price. For example, French has “libre” and “gratuit.” Not so English; there is a word “gratis” that refers unambiguously to price, but no common adjective that refers unambiguously to freedom. So if you are speaking another language, we suggest you translate “free” into your language to make it clearer. See our list of translations of the term “free software” into various other languages ([393](#)).

Free software is often more reliable than nonfree software.

Open Source Software

The term “open source” software is used by some people to mean more or less the same category as free software. It is not exactly the same class of software: they accept some licenses that we consider too restrictive, and there are free software licenses they have not accepted. However, the differences in extension of the category are small: nearly all free software is open

source, and nearly all open source software is free.

We prefer the term “free software” because it refers to freedom—something that the term “open source” does not do.

Public Domain Software

Public domain software is software that is not copyrighted. If the source code is in the public domain, that is a special case of noncopylefted free software, which means that some copies or modified versions may not be free at all.

In some cases, an executable program can be in the public domain but the source code is not available. This is not free software, because free software requires accessibility of source code. Meanwhile, most free software is not in the public domain; it is copyrighted, and the copyright holders have legally given permission for everyone to use it in freedom, using a free software license.

Sometimes people use the term “public domain” in a loose fashion to mean “free” or “available gratis.” However, “public domain” is a legal term and means, precisely, “not copyrighted.” For clarity, we recommend using “public domain” for that meaning only, and using other terms to convey the other meanings.

Under the Berne Convention, which most countries have signed, anything written down is automatically copyrighted. This includes programs. Therefore, if you want a program you have written to be in the public domain, you must take some legal steps to disclaim the

copyright on it; otherwise, the program is copyrighted.

Copylefted Software

Copylefted software is free software whose distribution terms ensure that all copies of all versions carry more or less the same distribution terms. This means, for instance, that copyleft licenses generally disallow others to add additional requirements to the software (though a limited set of safe added requirements can be allowed) and require making source code available. This shields the program, and its modified versions, from some of the common ways of making a program proprietary.

Some copyleft licenses, such as GPL version 3, block other means of turning software proprietary, such as tivoization.

In the GNU Project, we copyleft almost all the software we write, because our goal is to give *every* user the freedoms implied by the term “free software.” See the essay “Copyleft” ([195](#)) for more explanation of how copyleft works and why we use it.

Copyleft is a general concept; to copyleft an actual program, you need to use a specific set of distribution terms. There are many possible ways to write copyleft distribution terms, so in principle there can be many copyleft free software licenses. However, in actual practice nearly all copylefted software uses the GNU General Public License. Two different copyleft licenses are usually “incompatible,” which means it is illegal to merge the code using one license with the code using the other license; therefore, it is good for the community if people use a single copyleft license.

Noncopylefted Free Software

Noncopylefted free software comes from the author with permission to redistribute and modify, and also to add additional restrictions to it.

If a program is free but not copylefted, then some copies or modified versions may not be free at all. A software company can compile the program, with or without modifications, and distribute the executable file as a proprietary software product.

The X Window System illustrates this. The X Consortium releases X11 with distribution terms that make it noncopylefted free software. If you wish, you can get a copy which has those distribution terms and is free. However, there are nonfree versions as well, and there are (or at least were) popular workstations and PC graphics boards for which nonfree versions are the only ones that work. If you are using this hardware, X11 is not free software for you. The developers of X11 even made X11 nonfree for a while; they were able to do this because others had contributed their code under the same noncopyleft license.

Lax Permissive Licensed Software

Lax permissive licenses include the X11 license and the two BSD licenses. These licenses permit almost any use of the code, including distributing proprietary binaries with or without changing the source code.

GPL-Covered Software

The GNU GPL (General Public License) is one specific set of distribution terms for copying a program. The GNU Project uses it as the distribution terms for most GNU software.

To equate free software with GPL-covered software is therefore an error.

The GNU Operating System

The GNU operating system is the Unix-like operating system, which is entirely free software, that we in the GNU Project have developed since 1984.

A Unix-like operating system consists of many programs. The GNU system includes all the GNU software, as well as many other packages, such as the X Window System and \TeX , which are not GNU software.

The first test release of the complete GNU system was in 1996. This includes the GNU Hurd, our kernel, developed since 1990. In 2001 the GNU system (including the GNU Hurd) began working fairly reliably, but the Hurd still lacks some important features, so it is not widely used. Meanwhile, the GNU/Linux system, an offshoot of the GNU operating system which uses Linux as the kernel instead of the GNU Hurd, has been a great success since the 90s.

Since the purpose of GNU is to be free, every single component in the GNU operating system has to be free software. They don't all have to be copylefted, however; any kind of free software is legally suitable to include if it helps meet technical goals. And it isn't necessary for all the components to be GNU software, individually. GNU

can and does include noncopyrighted free software such as the X Window System that were developed by other projects.

GNU Programs

“GNU programs” is equivalent to GNU software. A program Foo is a GNU program if it is GNU software. We also sometimes say it is a “GNU package.”

GNU Software

GNU software is software that is released under the auspices of the GNU Project. If a program is GNU software, we also say that it is a GNU program or a GNU package. The README or manual of a GNU package should say it is one; also, the Free Software Directory identifies all GNU packages.

Most GNU software is copylefted, but not all; however, all GNU software must be free software.

Some GNU software was written by staff of the Free Software Foundation, but most GNU software comes from many volunteers. (Some of these volunteers are paid by companies or universities, but they are volunteers for us.) Some contributed software is copyrighted by the Free Software Foundation; some is copyrighted by the contributors who wrote it.

Nonfree Software

Nonfree software is any software that is not free. Its use, redistribution or modification is prohibited, or requires you to ask for permission, or is restricted so much that

you effectively can't do it freely.

Proprietary Software

Proprietary software is another name for nonfree software. In the past we subdivided nonfree software into “semifree software,” which could be modified and redistributed noncommercially, and “proprietary software,” which could not be. But we have dropped that distinction and now use “proprietary software” as synonymous with nonfree software.

The Free Software Foundation follows the rule that we cannot install any proprietary program on our computers except temporarily for the specific purpose of writing a free replacement for that very program. Aside from that, we feel there is no possible excuse for installing a proprietary program.

For example, we felt justified in installing Unix on our computer in the 1980s, because we were using it to write a free replacement for Unix. Nowadays, since free operating systems are available, the excuse is no longer applicable; we do not use any nonfree operating systems, and any new computer we install must run a completely free operating system.

We don't insist that users of GNU, or contributors to GNU, have to live by this rule. It is a rule we made for ourselves. But we hope you will follow it too, for your freedom's sake.

Freeware

The term “freeware” has no clear accepted definition, but it is commonly used for packages which permit

It is commonly used for packages which permit redistribution but not modification (and their source code is not available). These packages are *not* free software, so please don't use "freeware" to refer to free software.

Shareware

Shareware is software which comes with permission for people to redistribute copies, but says that anyone who continues to use a copy is *required* to pay a license fee.

Shareware is not free software, or even semifree. There are two reasons it is not:

- For most shareware, source code is not available; thus, you cannot modify the program at all.
- Shareware does not come with permission to make a copy and install it without paying a license fee, not even for individuals engaging in nonprofit activity. (In practice, people often disregard the distribution terms and do this anyway, but the terms don't permit it.)

Private Software

Private or custom software is software developed for one user (typically an organization or company). That user keeps it and uses it, and does not release it to the public either as source code or as binaries.

A private program is free software in a trivial sense if its sole user has full rights to it.

In general we do not believe it is wrong to develop a program and not release it. There are occasions when a program is so useful that withholding it from release is

programs or even the marketing themselves as treating humanity badly. However, most programs are not that important, so not releasing them is not particularly harmful. Thus, there is no conflict between the development of private or custom software and the principles of the free software movement.

Nearly all employment for programmers is in development of custom software; therefore most programming jobs are, or could be, done in a way compatible with the free software movement.

Commercial Software

Commercial software is software being developed by a business which aims to make money from the use of the software. “Commercial” and “proprietary” are not the same thing! Most commercial software is proprietary, but there is commercial free software, and there is noncommercial nonfree software.

For example, GNU Ada is developed by a company. It is always distributed under the terms of the GNU GPL, and every copy is free software; but its developers sell support contracts. When their salesmen speak to prospective customers, sometimes the customers say, “We would feel safer with a commercial compiler.” The salesmen reply, “GNU Ada *is* a commercial compiler; it happens to be free software.”

For the GNU Project, the emphasis is in the other order: the important thing is that GNU Ada is free software; whether it is commercial is just a detail. However, the additional development of GNU Ada that results from its being commercial is definitely beneficial.

Please help spread the awareness that free commercial software is possible. You can do this by making an effort not to say “commercial” when you mean “proprietary.”

Chapter 14

Why Open Source Misses the Point of Free Software

Copyright © 2007, 2008, 2010 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2007.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

When we call software “free,” we mean that it respects the users’ essential freedoms: the freedom to run it, to study and change it, and to redistribute copies with or without changes. This is a matter of freedom, not price, so think of “free speech,” not “free beer.”

These freedoms are vitally important. They are essential, not just for the individual users’ sake, but for society as a whole because they promote social solidarity—that is, sharing and cooperation. They become even

more important as our culture and life activities are increasingly digitized. In a world of digital sounds, images, and words, free software becomes increasingly essential for freedom in general.

Tens of millions of people around the world now use free software; the public schools of some regions of India and Spain now teach all students to use the free GNU/Linux operating system. Most of these users, however, have never heard of the ethical reasons for which we developed this system and built the free software community, because nowadays this system and community are more often spoken of as “open source,” attributing them to a different philosophy in which these freedoms are hardly mentioned.

The free software movement has campaigned for computer users’ freedom since 1983. In 1984 we launched the development of the free operating system GNU, so that we could avoid the nonfree operating systems that deny freedom to their users. During the 1980s, we developed most of the essential components of the system and designed the GNU General Public License (GNU GPL) to release them under—a license designed specifically to protect freedom for all users of a

program.

Not all of the users and developers of free software agreed with the goals of the free software movement. In 1998, a part of the free software community splintered off and began campaigning in the name of “open source.” The term was originally proposed to avoid a possible misunderstanding of the term “free software,” but it soon became associated with philosophical views quite different from those of the free software movement.

Some of the supporters of open source considered the term a “marketing campaign for free software,” which would appeal to business executives by highlighting the software’s practical benefits, while not raising issues of right and wrong that they might not like to hear. Other supporters flatly rejected the free software movement’s ethical and social values. Whichever their views, when campaigning for open source, they neither cited nor advocated those values. The term “open source” quickly became associated with ideas and arguments based only on practical values, such as making or having powerful, reliable software. Most of the supporters of open source have come to it since then, and they make the same association.

Nearly all open source software is free software. The two terms describe almost the same category of software, but they stand for views based on fundamentally different values. Open source is a development methodology; free software is a social movement. For the free software movement, free software is an ethical imperative, because only free software respects the users' freedom. By contrast, the philosophy of open source considers issues in terms of how to make software "better"—in a practical sense only. It says that nonfree software is an inferior solution to the practical problem at hand. For the free software movement, however, nonfree software is a social problem, and the solution is to stop using it and move to free software.

"Free software." "Open source." If it's the same software, does it matter which name you use? Yes, because different words convey different ideas. While a free program by any other name would give you the same freedom today, establishing freedom in a lasting way depends above all on teaching people to value freedom. If you want to help do this, it is essential to speak of "free software."

We in the free software movement don't think of the open source camp as an enemy; the enemy is proprietary (nonfree) software. But we want people to know we stand for freedom, so we do not accept being mislabeled as open source supporters.

Common Misunderstandings of “Free Software” and “Open Source”

The term “free software” is prone to misinterpretation: an unintended meaning, “software you can get for zero price,” fits the term just as well as the intended meaning, “software which gives the user certain freedoms.” We address this problem by publishing the definition of free software, and by saying, “Think of ‘free speech,’ not ‘free beer.’” This is not a perfect solution; it cannot completely eliminate the problem. An unambiguous and correct term would be better, if it didn't present other problems.

Unfortunately, all the alternatives in English have problems of their own. We've looked at many that people have suggested, but none is so clearly “right” that switching to it would be a good idea. (For instance, in

something to a head of a good idea (or license, in some contexts the French and Spanish word “libre” works well, but people in India do not recognize it at all.) Every proposed replacement for “free software” has some kind of semantic problem—and this includes “open source software.”

The official definition of “open source software” [\[1\]](#) (which is published by the Open Source Initiative and is too long to include here) was derived indirectly from our criteria for free software. It is not the same; it is a little looser in some respects, so the open source people have accepted a few licenses that we consider unacceptably restrictive. Also, they judge solely by the license of the source code, whereas our criterion also considers whether a device will let you *run* your modified version of the program. Nonetheless, their definition agrees with our definition in most cases.

However, the obvious meaning for the expression “open source software”—and the one most people seem to think it means—is “You can look at the source code.” That criterion is much weaker than the free software definition, much weaker also than the official definition of open source. It includes many programs that are neither free nor open source.

Since that obvious meaning for “open source” is not the meaning that its advocates intend, the result is that most people misunderstand the term. According to writer Neal Stephenson, “Linux is ‘open source’ software, meaning simply, anyone can get copies of its source code files.” [2] I don’t think he deliberately sought to reject or dispute the “official” definition. I think he simply applied the conventions of the English language to come up with a meaning for the term. The state of Kansas published a similar definition: “Make use of open-source software (OSS). OSS is software for which the source code is freely and publicly available, though the specific licensing agreements vary as to what one is allowed to do with that code.”

The *New York Times* has run an article that stretches the meaning of the term to refer to user beta testing [3]—letting a few users try an early version and give confidential feedback—which proprietary software developers have practiced for decades.

Open source supporters try to deal with this by pointing to their official definition, but that corrective approach is less effective for them than it is for us. The

term “free software” has two natural meanings, one of which is the intended meaning, so a person who has grasped the idea of “free speech, not free beer” will not get it wrong again. But the term “open source” has only one natural meaning, which is different from the meaning its supporters intend. So there is no succinct way to explain and justify its official definition. That makes for worse confusion.

Another misunderstanding of “open source” is the idea that it means “not using the GNU GPL.” This tends to accompany another misunderstanding that “free software” means “GPL-covered software.” These are both mistaken, since the GNU GPL qualifies as an open source license and most of the open source licenses qualify as free software licenses.

The term “open source” has been further stretched by its application to other activities, such as government, education, and science, where there is no such thing as source code, and where criteria for software licensing are simply not pertinent. The only thing these activities have in common is that they somehow invite people to participate. They stretch the term so far that it only means “participatory.”

Different Values Can Lead to Similar Conclusions...but Not Always

Radical groups in the 1960s had a reputation for factionalism: some organizations split because of disagreements on details of strategy, and the two daughter groups treated each other as enemies despite having similar basic goals and values. The right wing made much of this and used it to criticize the entire left.

Some try to disparage the free software movement by comparing our disagreement with open source to the disagreements of those radical groups. They have it backwards. We disagree with the open source camp on the basic goals and values, but their views and ours lead in many cases to the same practical behavior—such as developing free software.

As a result, people from the free software movement and the open source camp often work together on practical projects such as software development. It is remarkable that such different philosophical views can so often motivate different people to participate in the same projects. Nonetheless, there are situations where these

projects. Nonetheless, there are situations where these fundamentally different views lead to very different actions.

The idea of open source is that allowing users to change and redistribute the software will make it more powerful and reliable. But this is not guaranteed. Developers of proprietary software are not necessarily incompetent. Sometimes they produce a program that is powerful and reliable, even though it does not respect the users' freedom. Free software activists and open source enthusiasts will react very differently to that.

A pure open source enthusiast, one that is not at all influenced by the ideals of free software, will say, "I am surprised you were able to make the program work so well without using our development model, but you did. How can I get a copy?" This attitude will reward schemes that take away our freedom, leading to its loss.

The free software activist will say, "Your program is very attractive, but I value my freedom more. So I reject your program. Instead I will support a project to develop a free replacement." If we value our freedom, we can act to maintain and defend it.

Powerful, Reliable Software Can Be Bad

The idea that we want software to be powerful and reliable comes from the supposition that the software is designed to serve its users. If it is powerful and reliable, that means it serves them better.

But software can be said to serve its users only if it respects their freedom. What if the software is designed to put chains on its users? Then powerfulness means the chains are more constricting, and reliability that they are harder to remove. Malicious features, such as spying on the users, restricting the users, back doors, and imposed upgrades are common in proprietary software, and some open source supporters want to implement them in open source programs.

Under pressure from the movie and record companies, software for individuals to use is increasingly designed specifically to restrict them. This malicious feature is known as Digital Restrictions Management (DRM) (see <http://defectivebydesign.org>) and is the antithesis in spirit of the freedom that free software aims to provide. And not just in spirit: since the goal of DRM is to trample your freedom, DRM developers try

DRM is to ~~hamp~~ your freedom, DRM developers try to make it hard, impossible, or even illegal for you to change the software that implements the DRM.

Yet some open source supporters have proposed “open source DRM” software. Their idea is that, by publishing the source code of programs designed to restrict your access to encrypted media and by allowing others to change it, they will produce more powerful and reliable software for restricting users like you. The software would then be delivered to you in devices that do not allow you to change it.

This software might be open source and use the open source development model, but it won’t be free software since it won’t respect the freedom of the users that actually run it. If the open source development model succeeds in making this software more powerful and reliable for restricting you, that will make it even worse.

Fear of Freedom

The main initial motivation of those who split off the open source camp from the free software movement was that the ethical ideas of “free software” made some people uneasy. That’s true: raising ethical issues such as

uneasy. That's true. Raising ethical issues such as freedom, talking about responsibilities as well as convenience, is asking people to think about things they might prefer to ignore, such as whether their conduct is ethical. This can trigger discomfort, and some people may simply close their minds to it. It does not follow that we ought to stop talking about these issues.

That is, however, what the leaders of open source decided to do. They figured that by keeping quiet about ethics and freedom, and talking only about the immediate practical benefits of certain free software, they might be able to “sell” the software more effectively to certain users, especially business.

This approach has proved effective, in its own terms. The rhetoric of open source has convinced many businesses and individuals to use, and even develop, free software, which has extended our community—but only at the superficial, practical level. The philosophy of open source, with its purely practical values, impedes understanding of the deeper ideas of free software; it brings many people into our community, but does not teach them to defend it. That is good, as far as it goes, but it is not enough to make freedom secure. Attracting users to free software takes them just part of the way to

users to free software takes them just part of the way to becoming defenders of their own freedom.

Sooner or later these users will be invited to switch back to proprietary software for some practical advantage. Countless companies seek to offer such temptation, some even offering copies gratis. Why would users decline? Only if they have learned to value the freedom free software gives them, to value freedom in and of itself rather than the technical and practical convenience of specific free software. To spread this idea, we have to talk about freedom. A certain amount of the “keep quiet” approach to business can be useful for the community, but it is dangerous if it becomes so common that the love of freedom comes to seem like an eccentricity.

That dangerous situation is exactly what we have. Most people involved with free software, especially its distributors, say little about freedom—usually because they seek to be “more acceptable to business.” Nearly all GNU/Linux operating system distributions add proprietary packages to the basic free system, and they invite users to consider this an advantage rather than a flaw.

Proprietary add-on software and partially nonfree GNU/Linux distributions find fertile ground because most of our community does not insist on freedom with its software. This is no coincidence. Most GNU/Linux users were introduced to the system through “open source” discussion, which doesn’t say that freedom is a goal. The practices that don’t uphold freedom and the words that don’t talk about freedom go hand in hand, each promoting the other. To overcome this tendency, we need more, not less, talk about freedom.

Conclusion

As the advocates of open source draw new users into our community, we free software activists must shoulder the task of bringing the issue of freedom to their attention. We have to say, “It’s free software and it gives you freedom!”—more and louder than ever. Every time you say “free software” rather than “open source,” you help our campaign.

Notes

- Joe Barr’s article “Live and Let License”

(ITworld.com, 22 May 2001,

<http://www.itworld.com/LWD010523vcontrol4>)

gives his perspective on this issue.

- Karim R. Lakhani and Robert G. Wolf's paper on the motivation of free software developers ("Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects," in *Perspectives on Free and Open Source Software*, edited by J. Feller and others (Cambridge: MIT Press, 2005)) says that a considerable fraction are motivated by the view that software should be free. This is despite the fact that they surveyed the developers on SourceForge, a site that does not support the view that this is an ethical issue.

Endnotes

¹ See <http://opensource.org/docs/osd> for the full definition.

² Neal Stephenson, *In the Beginning... Was the Command Line* (New York: HarperCollins Publishers, 1999), p. 94.

³ Mary Jane Irwin, "The Brave New World of Open-Source Game Design," *New York Times*, online ed., 7 February 2009, <http://www.nytimes.com/2009/02/07/technology/07open.html>.

<http://www.nytimes.com/external/qigaom/2009/02/07/07/qigaom-the-brave-new-world-of-open-source-game-/design-37415.html>.

Chapter 15

Did You Say “Intellectual Property”? It’s a Seductive Mirage

Copyright © 2004, 2006, 2007, 2009, 2010 Richard Stallman

This article was written in 2004 and published in *Policy Futures in Education*, vol. 4, n. 4, pp. 334–336, 2006. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

It has become fashionable to toss copyright, patents, and trademarks—three separate and different entities involving three separate and different sets of laws—plus a dozen other laws into one pot and call it “intellectual property.” The distorting and confusing term did not become common by accident. Companies that gain from the confusion promoted it. The clearest way out of the confusion is to reject the term entirely.

CONCLUSION IS TO REJECT THE TERM ENTIRELY.

According to Professor Mark Lemley, now of the Stanford Law School, the widespread use of the term “intellectual property” is a fashion that followed the 1967 founding of the World “Intellectual Property” Organization (WIPO), and only became really common in recent years. (WIPO is formally a UN organization, but in fact represents the interests of the holders of copyrights, patents, and trademarks.)

The term carries a bias that is not hard to see: it suggests thinking about copyright, patents and trademarks by analogy with property rights for physical objects. (This analogy is at odds with the legal philosophies of copyright law, of patent law, and of trademark law, but only specialists know that.) These laws are in fact not much like physical property law, but use of this term leads legislators to change them to be more so. Since that is the change desired by the companies that exercise copyright, patent and trademark powers, the bias introduced by the term “intellectual property” suits them.

The bias is reason enough to reject the term, and people have often asked me to propose some other name for the overall category—or have proposed their own alternatives (often humorous). Suggestions include IMPs, for Imposed Monopoly Privileges, and GOLEMs, for Government-Originated Legally Enforced Monopolies. Some speak of “exclusive rights regimes,” but referring to restrictions as “rights” is doublethink too.

Some of these alternative names would be an improvement, but it is a mistake to replace “intellectual property” with any other term. A different name will not address the term’s deeper problem: overgeneralization. There is no such unified thing as “intellectual property”—it is a mirage. The only reason people think it makes sense as a coherent category is that widespread use of the term has misled them.

The term “intellectual property” is at best a catch-all to lump together disparate laws. Nonlawyers who hear one term applied to these various laws tend to assume they are based on a common principle and function similarly.

Nothing could be further from the case. These laws originated separately, evolved differently, cover different activities, have different rules, and raise different public policy issues.

Copyright law was designed to promote authorship and art, and covers the details of expression of a work. Patent law was intended to promote the publication of useful ideas, at the price of giving the one who publishes an idea a temporary monopoly over it—a price that may be worth paying in some fields and not in others.

Trademark law, by contrast, was not intended to promote any particular way of acting, but simply to enable buyers to know what they are buying. Legislators under the influence of the term “intellectual property,” however, have turned it into a scheme that provides incentives for advertising.

Since these laws developed independently, they are different in every detail, as well as in their basic purposes and methods. Thus, if you learn some fact about copyright law, you’d be wise to assume that patent law is

different. You'll rarely go wrong!

People often say "intellectual property" when they really mean some larger or smaller category. For instance, rich countries often impose unjust laws on poor countries to squeeze money out of them. Some of these laws are "intellectual property" laws, and others are not; nonetheless, critics of the practice often grab for that label because it has become familiar to them. By using it, they misrepresent the nature of the issue. It would be better to use an accurate term, such as "legislative colonization," that gets to the heart of the matter.

Laymen are not alone in being confused by this term. Even law professors who teach these laws are lured and distracted by the seductiveness of the term "intellectual property," and make general statements that conflict with facts they know. For example, one professor wrote in 2006:

Unlike their descendants who now work the floor at WIPO, the framers of the US constitution had a principled, procompetitive attitude to intellectual property. They knew rights might be necessary, but...they tied congress's hands, restricting its power in

multiple ways.

That statement refers to Article I, Section 8, Clause 8, of the US Constitution, which authorizes copyright law and patent law. That clause, though, has nothing to do with trademark law or various others. The term “intellectual property” led that professor to make false generalization.

The term “intellectual property” also leads to simplistic thinking. It leads people to focus on the meager commonality in form that these disparate laws have—that they create artificial privileges for certain parties—and to disregard the details which form their substance: the specific restrictions each law places on the public, and the consequences that result. This simplistic focus on the form encourages an “economistic” approach to all these issues.

Economics operates here, as it often does, as a vehicle for unexamined assumptions. These include assumptions about values, such as that amount of production matters while freedom and way of life do not, and factual assumptions which are mostly false, such as

that copyrights on music supports musicians, or that patents on drugs support life-saving research.

Another problem is that, at the broad scale implicit in the term “intellectual property,” the specific issues raised by the various laws become nearly invisible. These issues arise from the specifics of each law—precisely what the term “intellectual property” encourages people to ignore. For instance, one issue relating to copyright law is whether music sharing should be allowed; patent law has nothing to do with this. Patent law raises issues such as whether poor countries should be allowed to produce life-saving drugs and sell them cheaply to save lives; copyright law has nothing to do with such matters.

Neither of these issues is solely economic in nature, and their noneconomic aspects are very different; using the shallow economic overgeneralization as the basis for considering them means ignoring the differences. Putting the two laws in the “intellectual property” pot obstructs clear thinking about each one.

Thus, any opinions about “the issue of intellectual property,” and any generalizations about this supposed

property and any generalizations about this supposed category are almost surely foolish. If you think all those laws are one issue, you will tend to choose your opinions from a selection of sweeping overgeneralizations, none of which is any good.

If you want to think clearly about the issues raised by patents, or copyrights, or trademarks, or various other different laws, the first step is to forget the idea of lumping them together, and treat them as separate topics. The second step is to reject the narrow perspectives and simplistic picture the term “intellectual property” suggests. Consider each of these issues separately, in its fullness, and you have a chance of considering them well.

And when it comes to reforming WIPO, among other things let's call for changing its name.

Chapter 16

Words to Avoid (or Use with Care)

Because They Are Loaded or Confusing

Copyright © 1996, 1997, 1998, 1999, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Free Software Foundation, Inc.

This list was first published on <http://gnu.org>, in 1996. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

There are a number of words and phrases that we recommend avoiding, or avoiding in certain contexts and usages. Some are ambiguous or misleading; others presuppose a viewpoint that we hope you disagree with. (See also “Categories of Free and Nonfree Software,” on [\(119\)](#).)

BSD-Style

The expression “BSD-style license” leads to confusion because it lumps together licenses that have important differences. For instance, the original BSD license with the advertising clause is incompatible with the GNU General Public License, but the revised BSD license is compatible with the GPL.

To avoid confusion, it is best to name the specific license in question and avoid the vague term “BSD-style.”

Closed

Describing nonfree software as “closed” clearly refers to the term “open source.” In the free software movement, we do not want to be confused with the open source camp, so we are careful to avoid saying things that would encourage people to lump us in with them. For instance, we avoid describing nonfree software as “closed.” We call it “nonfree” or “proprietary.”

Cloud Computing

The term “cloud computing” is a marketing buzzword with no clear meaning. It is used for a range of different

activities whose only common characteristic is that they use the Internet for something beyond transmitting files. Thus, the term is a nexus of confusion. If you base your thinking on it, your thinking will be vague.

When thinking about or responding to a statement someone else has made using this term, the first step is to clarify the topic. Which kind of activity is the statement really about, and what is a good, clear term for that activity? Once the topic is clear, the discussion can head for a useful conclusion.

Curiously, Larry Ellison, a proprietary software developer, also noted the vacuity of the term “cloud computing.” [\[1\]](#) He decided to use the term anyway because, as a proprietary software developer, he isn’t motivated by the same ideals as we are.

Commercial

Please don’t use “commercial” as a synonym for “nonfree.” That confuses two entirely different issues.

A program is commercial if it is developed as a business activity. A commercial program can be free or

nonfree, depending on its manner of distribution. Likewise, a program developed by a school or an individual can be free or nonfree, depending on its manner of distribution. The two questions—what sort of entity developed the program and what freedom its users have—are independent.

In the first decade of the free software movement, free software packages were almost always noncommercial; the components of the GNU/Linux operating system were developed by individuals or by nonprofit organizations such as the FSF and universities. Later, in the 1990s, free commercial software started to appear.

Free commercial software is a contribution to our community, so we should encourage it. But people who think that “commercial” means “nonfree” will tend to think that the “free commercial” combination is self-contradictory, and dismiss the possibility. Let’s be careful not to use the word “commercial” in that way.

Compensation

To speak of “compensation for authors” in connection

with copyright carries the assumptions that (1) copyright exists for the sake of authors and (2) whenever we read something, we take on a debt to the author which we must then repay. The first assumption is simply false, and the second is outrageous.

Consumer

The term “consumer,” when used to refer to computer users, is loaded with assumptions we should reject. Playing a digital recording, or running a program, does not consume it.

The terms “producer” and “consumer” come from economic theory, and bring with them its narrow perspective and misguided assumptions. They tend to warp your thinking.

In addition, describing the users of software as “consumers” presumes a narrow role for them: it regards them as cattle that passively graze on what others make available to them.

This kind of thinking leads to travesties like the CBDTPA, the “Consumer Broadband and Digital

Television Promotion Act," which would require copying restriction facilities in every digital device. If all the users do is "consume," then why should they mind?

The shallow economic conception of users as "consumers" tends to go hand in hand with the idea that published works are mere "content."

To describe people who are not limited to passive use of works, we suggest terms such as "individuals" and "citizens."

Content

If you want to describe a feeling of comfort and satisfaction, by all means say you are "content," but using the word as a noun to describe written and other works of authorship adopts an attitude you might rather avoid. It regards these works as a commodity whose purpose is to fill a box and make money. In effect, it disparages the works themselves.

Those who use this term are often the publishers that push for increased copyright power in the name of the authors ("creators," as they say) of the works. The term "content" reveals their real attitude towards these works

Content reveals their real attitude towards these works and their authors. (See Courtney Love's open letter to Steve Case [\[2\]](#) and search for "content provider" in that page. Alas, Ms. Love is unaware that the term "intellectual property" is also biased and confusing.)

However, as long as other people use the term "content provider," political dissidents can well call themselves "malcontent providers."

The term "content management" takes the prize for vacuity. "Content" means "some sort of information," and "management" in this context means "doing something with it." So a "content management system" is a system for doing something to some sort of information. Nearly all programs fit that description.

In most cases, that term really refers to a system for updating pages on a web site. For that, we recommend the term "web site revision system" (WRS).

Creator

The term "creator" as applied to authors implicitly compares them to a deity ("the creator"). The term is used by publishers to elevate authors' moral standing

used by publishers to give their authors more standing above that of ordinary people in order to justify giving them increased copyright power, which the publishers can then exercise in their name. We recommend saying “author” instead. However, in many cases “copyright holder” is what you really mean.

Digital Goods

The term “digital goods,” as applied to copies of works of authorship, erroneously identifies them with physical goods—which cannot be copied, and which therefore have to be manufactured and sold.

Digital Rights Management

“Digital Rights Management” refers to technical schemes designed to impose restrictions on computer users. The use of the word “rights” in this term is propaganda, designed to lead you unawares into seeing the issue from the viewpoint of the few that impose the restrictions, and ignoring that of the general public on whom these restrictions are imposed.

Good alternatives include “Digital Restrictions Management ” and “digital handcuffs ”

Ecosystem

It is a mistake to describe the free software community, or any human community, as an “ecosystem,” because that word implies the absence of ethical judgment.

The term “ecosystem” implicitly suggests an attitude of nonjudgmental observation: don’t ask how what *should* happen, just study and explain what *does* happen. In an ecosystem, some organisms consume other organisms. We do not ask whether it is fair for an owl to eat a mouse or for a mouse to eat a plant, we only observe that they do so. Species’ populations grow or shrink according to the conditions; this is neither right nor wrong, merely an ecological phenomenon.

By contrast, beings that adopt an ethical stance towards their surroundings can decide to preserve things that, on their own, might vanish—such as civil society, democracy, human rights, peace, public health, clean air and water, endangered species, traditional arts...and computer users’ freedom.

For Free

If you want to say that a program is free software, please don't say that it is available "for free." That term specifically means "for zero price." Free software is a matter of freedom, not price.

Free software copies are often available for free—for example, by downloading via FTP. But free software copies are also available for a price on CD-ROMs; meanwhile, proprietary software copies are occasionally available for free in promotions, and some proprietary packages are normally available at no charge to certain users.

To avoid confusion, you can say that the program is available "as free software."

Freely Available

Don't use "freely available software" as a synonym for "free software." The terms are not equivalent. Software is "freely available" if anyone can easily get a copy. "Free software" is defined in terms of the freedom of users that have a copy of it. These are answers to different questions.

Freeware

Please don't use the term "freeware" as a synonym for "free software." The term "freeware" was used often in the 1980s for programs released only as executables, with source code not available. Today it has no particular agreed-on definition.

When using languages other than English, please avoid borrowing English terms such as "free software" or "freeware." It is better to translate the term "free software" into your language. (Please see [\(393\)](#) for a list of recommended unambiguous translations for the term "free software" into various languages.)

By using a word in your own language, you show that you are really referring to freedom and not just parroting some mysterious foreign marketing concept. The reference to freedom may at first seem strange or disturbing to your compatriots, but once they see that it means exactly what it says, they will really understand what the issue is.

Give Away Software

It's misleading to use the term "give away" to mean "distribute a program as free software." This locution has the same problem as "for free": it implies the issue is price, not freedom. One way to avoid the confusion is to say "release as free software."

Hacker

A hacker is someone who enjoys playful cleverness [3]—not necessarily with computers. The programmers in the old MIT free software community of the 60s and 70s referred to themselves as hackers. Around 1980, journalists who discovered the hacker community mistakenly took the term to mean "security breaker."

Please don't spread this mistake. People who break security are "crackers."

Intellectual Property

Publishers and lawyers like to describe copyright as "intellectual property"—a term also applied to patents, trademarks, and other more obscure areas of law. These laws have so little in common and differ so much that it

and have to live in comfort, and other so many, that it is ill-advised to generalize about them. It is best to talk specifically about “copyright,” or about “patents,” or about “trademarks.”

The term “intellectual property” carries a hidden assumption—that the way to think about all these disparate issues is based on an analogy with physical objects, and our conception of them as physical property.

When it comes to copying, this analogy disregards the crucial difference between material objects and information: information can be copied and shared almost effortlessly, while material objects can’t be.

To avoid spreading unnecessary bias and confusion, it is best to adopt a firm policy not to speak or even think in terms of “intellectual property.”

The hypocrisy of calling these powers “rights” is starting to make the World “Intellectual Property” Organization embarrassed.

LAMP System

“LAMP” stands for “Linux, Apache, MySQL and PHP”—a common combination of software to use on a web server, except that “Linux” in this context really refers to the GNU/Linux system. So instead of “LAMP” it should be “GLAMP”: “GNU, Linux, Apache, MySQL and PHP.”

Linux System

Linux is the name of the kernel that Linus Torvalds developed starting in 1991. The operating system in which Linux is used is basically GNU with Linux added. To call the whole system “Linux” is both unfair and confusing. Please call the complete system GNU/Linux, both to give the GNU Project credit and to distinguish the whole system from the kernel alone.

Market

It is misleading to describe the users of free software, or the software users in general, as a “market.”

This is not to say there is no room for markets in the free software community. If you have a free software support business, then you have clients, and you trade

with them in a market. As long as you respect their freedom, we wish you success in your market.

But the free software movement is a social movement, not a business, and the success it aims for is not a market success. We are trying to serve the public by giving it freedom—not competing to draw business away from a rival. To equate this campaign for freedom to a business’ efforts for mere success is to deny the importance of freedom and legitimize proprietary software.

MP3 Player

In the late 1990s it became feasible to make portable, solid-state digital audio players. Most support the patented MP3 codec, but not all. Some support the patent-free audio codecs Ogg Vorbis and FLAC, and may not even support MP3-encoded files at all, precisely to avoid these patents. To call such players “MP3 players” is not only confusing, it also puts MP3 in an undeserved position of privilege which encourages people to continue using that vulnerable format. We suggest the terms “digital audio player,” or simply “audio player” if context permits.

Open

Please avoid using the term “open” or “open source” as a substitute for “free software.” Those terms refer to a different position based on different values. Free software is a political movement; open source is a development model.

When referring to the open source position, using its name is appropriate; but please do not use it to label us or our work—that leads people to think we share those views.

PC

It’s OK to use the abbreviation “PC” to refer to a certain kind of computer hardware, but please don’t use it with the implication that the computer is running Microsoft Windows. If you install GNU/Linux on the same computer, it is still a PC.

The term “WC” has been suggested for a computer running Windows.

Photoshop

Please avoid using the term “photoshop” as a verb, meaning any kind of photo manipulation or image editing in general. Photoshop is just the name of one particular image editing program, which should be avoided since it is proprietary. There are plenty of free alternatives, such as GIMP.

Piracy

Publishers often refer to copying they don't approve of as “piracy.” In this way, they imply that it is ethically equivalent to attacking ships on the high seas, kidnapping and murdering the people on them. Based on such propaganda, they have procured laws in most of the world to forbid copying in most (or sometimes all) circumstances. (They are still pressuring to make these prohibitions more complete.)

If you don't believe that copying not approved by the publisher is just like kidnapping and murder, you might prefer not to use the word “piracy” to describe it. Neutral terms such as “unauthorized copying” (or “prohibited copying” for the situation where it is illegal)

are available for use instead. Some of us might even prefer to use a positive term such as “sharing information with your neighbor.”

PowerPoint

Please avoid using the term “PowerPoint” to mean any kind of slide presentation. “PowerPoint” is just the name of one particular proprietary program to make presentations, and there are plenty of free alternatives, such as T_EX’s `beamer` class and OpenOffice.org’s Impress.

Protection

Publishers’ lawyers love to use the term “protection” to describe copyright. This word carries the implication of preventing destruction or suffering; therefore, it encourages people to identify with the owner and publisher who benefit from copyright, rather than with the users who are restricted by it.

It is easy to avoid “protection” and use neutral terms instead. For example, instead of saying, “Copyright protection lasts a very long time” you can say

protection last a very long time, you can say,
“Copyright lasts a very long time.”

If you want to criticize copyright instead of supporting it, you can use the term “copyright restrictions.” Thus, you can say, “Copyright restrictions last a very long time.”

The term “protection” is also used to describe malicious features. For instance, “copy protection” is a feature that interferes with copying. From the user’s point of view, this is obstruction. So we could call that malicious feature “copy obstruction.” More often it is called Digital Restrictions Management (DRM)—see the Defective by Design campaign, at <http://www.defectivebydesign.org>.

RAND (Reasonable and Non-Discriminatory)

Standards bodies that promulgate patent-restricted standards that prohibit free software typically have a policy of obtaining patent licenses that require a fixed fee per copy of a conforming program. They often refer to such licenses by the term “RAND,” which stands for “reasonable and non-discriminatory.”

That term whitewashes a class of patent licenses that are normally neither reasonable nor nondiscriminatory. It is true that these licenses do not discriminate against any specific person, but they do discriminate against the free software community, and that makes them unreasonable. Thus, half of the term “RAND” is deceptive and the other half is prejudiced.

Standards bodies should recognize that these licenses are discriminatory, and drop the use of the term “reasonable and non-discriminatory” or “RAND” to describe them. Until they do so, writers who do not wish to join in the whitewashing would do well to reject that term. To accept and use it merely because patent-wielding companies have made it widespread is to let those companies dictate the views you express.

We suggest the term “uniform fee only,” or “UFO” for short, as a replacement. It is accurate because the only condition in these licenses is a uniform royalty fee.

Sell Software

The term “sell software” is ambiguous. Strictly speaking,

exchanging a copy of a free program for a sum of money is selling; but people usually associate the term “sell” with proprietary restrictions on the subsequent use of the software. You can be more precise, and prevent confusion, by saying either “distributing copies of a program for a fee” or “imposing proprietary restrictions on the use of a program,” depending on what you mean.

See “Selling Free Software” ([97](#)) for further discussion of this issue.

Software Industry

The term “software industry” encourages people to imagine that software is always developed by a sort of factory and then delivered to “consumers.” The free software community shows this is not the case. Software businesses exist, and various businesses develop free and/or nonfree software, but those that develop free software are not run like factories.

The term “industry” is being used as propaganda by advocates of software patents. They call software development “industry” and then try to argue that this means it should be subject to patent monopolies. The

European Parliament, rejecting software patents in 2003, [\[4\]](#) voted to define “industry” as “automated production of material goods.”

Theft

Copyright apologists often use words like “stolen” and “theft” to describe copyright infringement. At the same time, they ask us to treat the legal system as an authority on ethics: if copying is forbidden, it must be wrong.

So it is pertinent to mention that the legal system—at least in the US—rejects the idea that copyright infringement is “theft.” Copyright apologists are making an appeal to authority...and misrepresenting what authority says.

The idea that laws decide what is right or wrong is mistaken in general. Laws are, at their best, an attempt to achieve justice; to say that laws define justice or ethical conduct is turning things upside down.

Trusted Computing

“Trusted computing” is the proponents’ name for a

scheme to redesign computers so that application developers can trust your computer to obey them instead of you. From their point of view, it is “trusted”; from your point of view, it is “treacherous.”

Vendor

Please don't use the term “vendor” to refer generally to anyone that develops or packages software. Many programs are developed in order to sell copies, and their developers are therefore their vendors; this even includes some free software packages. However, many programs are developed by volunteers or organizations which do not intend to sell copies. These developers are not vendors. Likewise, only some of the packagers of GNU/Linux distributions are vendors. We recommend the general term “supplier” instead.

Endnotes

1 Dan Farber, “Oracle’s Ellison Nails Cloud Computing” 26 September 2008, <http://news.cnet.com/8301-13953\ 3-10052188-80.html>.

2 An unedited transcript of American rock musician Courtney

Love's 16 May 2000 speech to the Digital Hollywood online-entertainment conference, in New York, is available at <http://salon.com/technology/feature/2000/06/14/love/print.html>.

3 See my article, "On Hacking" at <http://stallman.org/articles/on-hacking.html>.

4 "Directive on the patentability of computer-implemented inventions," 24 September 2003, <http://eupat.ffii.org/papers/euoparl0309>.

Part III

Copyright, Copyleft

Chapter 17

The Right to Read: A Dystopian Short Story

Copyright © 1996, 2002, 2007, 2009, 2010 Richard Stallman

This essay was written in 1996 and was published in *Communications of the ACM*, vol. 40, n. 2, February 1997. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

From The Road to Tycho, a collection of articles about the antecedents of the Lunarian Revolution, published in Luna City in 2096.

For Dan Halbert, the road to Tycho began in college—when Lissa Lenz asked to borrow his computer. Hers had broken down, and unless she could borrow another, she would fail her midterm project. There was no one

she dared ask, except Dan.

This put Dan in a dilemma. He had to help her—but if he lent her his computer, she might read his books. Aside from the fact that you could go to prison for many years for letting someone else read your books, the very idea shocked him at first. Like everyone, he had been taught since elementary school that sharing books was nasty and wrong—something that only pirates would do.

And there wasn't much chance that the SPA—the Software Protection Authority—would fail to catch him. In his software class, Dan had learned that each book had a copyright monitor that reported when and where it was read, and by whom, to Central Licensing. (They used this information to catch reading pirates, but also to sell personal interest profiles to retailers.) The next time his computer was networked, Central Licensing would find out. He, as computer owner, would receive the harshest punishment—for not taking pains to prevent the crime.

Of course, Lissa did not necessarily intend to read

his books. She might want the computer only to write her midterm. But Dan knew she came from a middle-class family and could hardly afford the tuition, let alone her reading fees. Reading his books might be the only way she could graduate. He understood this situation; he himself had had to borrow to pay for all the research papers he read. (Ten percent of those fees went to the researchers who wrote the papers; since Dan aimed for an academic career, he could hope that his own research papers, if frequently referenced, would bring in enough to repay this loan.)

Later on, Dan would learn there was a time when anyone could go to the library and read journal articles, and even books, without having to pay. There were independent scholars who read thousands of pages without government library grants. But in the 1990s, both commercial and nonprofit journal publishers had begun charging fees for access. By 2047, libraries offering free public access to scholarly literature were a dim memory.

There were ways, of course, to get around the SPA and Central Licensing. They were themselves illegal. Dan had had a classmate in software. Frank Martucci, who

had obtained an illicit debugging tool, and used it to skip over the copyright monitor code when reading books. But he had told too many friends about it, and one of them turned him in to the SPA for a reward (students deep in debt were easily tempted into betrayal). In 2047, Frank was in prison, not for pirate reading, but for possessing a debugger.

Dan would later learn that there was a time when anyone could have debugging tools. There were even free debugging tools available on CD or downloadable over the net. But ordinary users started using them to bypass copyright monitors, and eventually a judge ruled that this had become their principal use in actual practice. This meant they were illegal; the debuggers' developers were sent to prison.

Programmers still needed debugging tools, of course, but debugger vendors in 2047 distributed numbered copies only, and only to officially licensed and bonded programmers. The debugger Dan used in software class was kept behind a special firewall so that it could be used only for class exercises.

It was also possible to bypass the copyright monitors by installing a modified system kernel. Dan would eventually find out about the free kernels, even entire free operating systems, that had existed around the turn of the century. But not only were they illegal, like debuggers—you could not install one if you had one, without knowing your computer's root password. And neither the FBI nor Microsoft Support would tell you that.

Dan concluded that he couldn't simply lend Lissa his computer. But he couldn't refuse to help her, because he loved her. Every chance to speak with her filled him with delight. And that she chose him to ask for help, that could mean she loved him too.

Dan resolved the dilemma by doing something even more unthinkable—he lent her the computer, and told her his password. This way, if Lissa read his books, Central Licensing would think he was reading them. It was still a crime, but the SPA would not automatically find out about it. They would only find out if Lissa reported him.

Of course, if the school ever found out that he had

given Lissa his own password, it would be curtains for both of them as students, regardless of what she had used it for. School policy was that any interference with their means of monitoring students' computer use was grounds for disciplinary action. It didn't matter whether you did anything harmful—the offense was making it hard for the administrators to check on you. They assumed this meant you were doing something else forbidden, and they did not need to know what it was.

Students were not usually expelled for this—not directly. Instead they were banned from the school computer systems, and would inevitably fail all their classes.

Later, Dan would learn that this kind of university policy started only in the 1980s, when university students in large numbers began using computers. Previously, universities maintained a different approach to student discipline; they punished activities that were harmful, not those that merely raised suspicion.

Lissa did not report Dan to the SPA. His decision to help her led to their marriage, and also led them to

help her led to their marriage, and also led them to question what they had been taught about piracy as children. The couple began reading about the history of copyright, about the Soviet Union and its restrictions on copying, and even the original United States Constitution. They moved to Luna, where they found others who had likewise gravitated away from the long arm of the SPA. When the Tycho Uprising began in 2062, the universal right to read soon became one of its central aims.

Author's Note [\[1\]](#)

The right to read is a battle being fought today. Although it may take 50 years for our present way of life to fade into obscurity, most of the specific laws and practices described above have already been proposed; many have been enacted into law in the US and elsewhere. In the US, the 1998 Digital Millennium Copyright Act (DMCA) established the legal basis to restrict the reading and lending of computerized books (and other works as well). The European Union imposed similar restrictions in a 2001 copyright directive. In France, under the DADVSI law adopted in 2006, mere possession of a copy of DeCSS – the free program to

possession of a copy of DECSS, the free program to decrypt video on a DVD, is a crime.

In 2001, Disney-funded Senator Hollings proposed a bill called the SSSCA that would require every new computer to have mandatory copy-restriction facilities that the user cannot bypass. Following the Clipper chip and similar US government key-escrow proposals, this shows a long-term trend: computer systems are increasingly set up to give absentees with clout control over the people actually using the computer system. The SSSCA was later renamed to the unpronounceable CBDTPA, which was glossed as the “Consume But Don’t Try Programming Act.”

The Republicans took control of the US senate shortly thereafter. They are less tied to Hollywood than the Democrats, so they did not press these proposals. Now that the Democrats are back in control, the danger is once again higher.

In 2001 the US began attempting to use the proposed Free Trade Area of the Americas (FTAA) treaty to impose the same rules on all the countries in the

Western Hemisphere. The FTAA is one of the so-called free trade treaties, which are actually designed to give business increased power over democratic governments; imposing laws like the DMCA is typical of this spirit. The FTAA was effectively killed by Lula, President of Brazil, who rejected the DMCA requirement and others.

@righttoreadauthorsnote

Since then, the US has imposed similar requirements on countries such as Australia and Mexico through bilateral “free trade” agreements, and on countries such as Costa Rica through another treaty, CAFTA. Ecuador’s President Correa refused to sign a “free trade” agreement with the US, but I’ve heard Ecuador had adopted something like the DMCA in 2003.

One of the ideas in the story was not proposed in reality until 2002. This is the idea that the FBI and Microsoft will keep the root passwords for your personal computers, and not let you have them.

The proponents of this scheme have given it names such as “trusted computing” and “Palladium.” We call it “treacherous computing” because the effect is to make

your computer obey companies even to the extent of disobeying and defying you. This was implemented in 2007 as part of Windows Vista; we expect Apple to do something similar. In this scheme, it is the manufacturer that keeps the secret code, but the FBI would have little trouble getting it.

What Microsoft keeps is not exactly a password in the traditional sense; no person ever types it on a terminal. Rather, it is a signature and encryption key that corresponds to a second key stored in your computer. This enables Microsoft, and potentially any web sites that cooperate with Microsoft, the ultimate control over what the user can do on his own computer.

Vista also gives Microsoft additional powers; for instance, Microsoft can forcibly install upgrades, and it can order all machines running Vista to refuse to run a certain device driver. The main purpose of Vista's many restrictions is to impose DRM (Digital Restrictions Management) that users can't overcome. The threat of DRM is why we have established the Defective by Design campaign.

When this story was first written, the SPA was threatening small Internet service providers, demanding they permit the SPA to monitor all users. Most ISPs surrendered when threatened, because they cannot afford to fight back in court. One ISP, Community ConneXion in Oakland, California, refused the demand and was actually sued. The SPA later dropped the suit, but obtained the DMCA, which gave them the power they sought.

The SPA, which actually stands for Software Publishers Association, has been replaced in its police-like role by the Business Software Alliance. The BSA is not, today, an official police force; unofficially, it acts like one. Using methods reminiscent of the erstwhile Soviet Union, it invites people to inform on their coworkers and friends. A BSA terror campaign in Argentina in 2001 made slightly veiled threats that people sharing software would be raped.

The university security policies described above are not imaginary. For example, a computer at one Chicago-area university displayed this message upon login:

This system is for the use of authorized users only. Individuals using this computer system without authority or in the excess of their authority are subject to having all their activities on this system monitored and recorded by system personnel. In the course of monitoring individuals improperly using this system or in the course of system maintenance, the activities of authorized user may also be monitored. Anyone using this system expressly consents to such monitoring and is advised that if such monitoring reveals possible evidence of illegal activity or violation of University regulations system personnel may provide the evidence of such monitoring to University authorities and/or law enforcement officials.

This is an interesting approach to the Fourth Amendment: pressure most everyone to agree, in advance, to waive their rights under it.

References

- United States Patent and Trademark Office, *Intellectual Property [sic] and the National Information Infrastructure: The Report of the Working Group on Intellectual Property [sic]*

Rights, Washington, DC: GPO, 1995.

- Samuelson, Pamela, “The Copyright Grab,” *Wired*, January 1996, n. 4.01.
- Boyle, James, “Sold Out,” *New York Times*, 31 March 1996, sec. 4, p. 15.
- Editorial, *Washington Post*, “Public Data or Private Data,” 3 November 1996, sec. C, p. 6.
- Union for the Public Domain—an organization that aims to resist and reverse the overextension of copyright and patent powers.

Endnotes

[1](#) This note has been updated several times since the first publication of the story.

Chapter 18

Misinterpreting Copyright—A Series of Errors

Copyright © 2002, 2003, 2007, 2009, 2010 Free Software Foundation, Inc.

This essay was first published on <http://gnu.org>, in 2002. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Something strange and dangerous is happening in copyright law. Under the US Constitution, copyright exists to benefit users—those who read books, listen to music, watch movies, or run software—not for the sake of publishers or authors. Yet even as people tend increasingly to reject and disobey the copyright restrictions imposed on them “for their own benefit,” the US government is adding more restrictions and trying to

US government is adding more restrictions, and trying to frighten the public into obedience with harsh new penalties.

How did copyright policies come to be diametrically opposed to their stated purpose? And how can we bring them back into alignment with that purpose? To understand, we should start by looking at the root of United States copyright law: the US Constitution.

Copyright in the US Constitution

When the US Constitution was drafted, the idea that authors were entitled to a copyright monopoly was proposed—and rejected. The founders of our country adopted a different premise, that copyright is not a natural right of authors, but an artificial concession made to them for the sake of progress. The Constitution gives permission for a copyright system with this clause (Article I, Section 8, Clause 8):

[Congress shall have the power] to promote the Progress of Science and the useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective

The Supreme Court has repeatedly affirmed that promoting progress means benefit for the users of copyrighted works. For example, in *Fox Film v. Doyal*, [\[1\]](#) the court said,

The sole interest of the United States and the primary object in conferring the [copyright] monopoly lie in the general benefits derived by the public from the labors of authors.

This fundamental decision explains why copyright is not *required* by the Constitution, only *permitted* as an option—and why it is supposed to last for “limited times.” If copyright were a natural right, something that authors have because they deserve it, nothing could justify terminating this right after a certain period of time, any more than everyone’s house should become public property after a certain lapse of time from its construction.

The “Copyright Bargain”

The copyright system works by providing privileges and

thus benefits to publishers and authors; but it does not do this for their sake. Rather, it does this to modify their behavior: to provide an incentive for authors to write more and publish more. In effect, the government spends the public's natural rights, on the public's behalf, as part of a deal to bring the public more published works. Legal scholars call this concept the "copyright bargain." It is like a government purchase of a highway or an airplane using taxpayers' money, except that the government spends our freedom instead of our money.

But is the bargain as it exists actually a good deal for the public? Many alternative bargains are possible; which one is best? Every issue of copyright policy is part of this question. If we misunderstand the nature of the question, we will tend to decide the issues badly.

The Constitution authorizes granting copyright powers to authors. In practice, authors typically cede them to publishers; it is usually the publishers, not the authors, who exercise these powers and get most of the benefits, though authors may get a small portion. Thus it is usually the publishers that lobby to increase copyright powers. To better reflect the reality of copyright rather

than the myth, this article refers to publishers rather than authors as the holders of copyright powers. It also refers to the users of copyrighted works as “readers,” even though using them does not always mean reading, because “the users” is remote and abstract.

The First Error: “Striking a Balance”

The copyright bargain places the public first: benefit for the reading public is an end in itself; benefits (if any) for publishers are just a means toward that end. Readers’ interests and publishers’ interests are thus qualitatively unequal in priority. The first step in misinterpreting the purpose of copyright is the elevation of the publishers to the same level of importance as the readers.

It is often said that US copyright law is meant to “strike a balance” between the interests of publishers and readers. Those who cite this interpretation present it as a restatement of the basic position stated in the Constitution; in other words, it is supposed to be equivalent to the copyright bargain.

But the two interpretations are far from equivalent; they are different conceptually, and different in their implications. The balance concept assumes that the readers' and publishers' interests differ in importance only quantitatively, in *how much weight* we should give them, and in what actions they apply to. The term "stakeholders" is often used to frame the issue in this way; it assumes that all kinds of interest in a policy decision are equally important. This view rejects the qualitative distinction between the readers' and publishers' interests which is at the root of the government's participation in the copyright bargain.

The consequences of this alteration are far-reaching, because the great protection for the public in the copyright bargain—the idea that copyright privileges can be justified only in the name of the readers, never in the name of the publishers—is discarded by the "balance" interpretation. Since the interest of the publishers is regarded as an end in itself, it can justify copyright privileges; in other words, the "balance" concept says that privileges can be justified in the name of someone other than the public.

As a practical matter, the consequence of the “balance” concept is to reverse the burden of justification for changes in copyright law. The copyright bargain places the burden on the publishers to convince the readers to cede certain freedoms. The concept of balance reverses this burden, practically speaking, because there is generally no doubt that publishers will benefit from additional privilege. Unless harm to the readers can be proved, sufficient to “outweigh” this benefit, we are led to conclude that the publishers are entitled to almost any privilege they request.

Since the idea of “striking a balance” between publishers and readers denies the readers the primacy they are entitled to, we must reject it.

Balancing against What?

When the government buys something for the public, it acts on behalf of the public; its responsibility is to obtain the best possible deal—best for the public, not for the other party in the agreement.

For example, when signing contracts with construction companies to build highways, the government aims to spend as little as possible of the public's money. Government agencies use competitive bidding to push the price down.

As a practical matter, the price cannot be zero, because contractors will not bid that low. Although not entitled to special consideration, they have the usual rights of citizens in a free society, including the right to refuse disadvantageous contracts; even the lowest bid will be high enough for some contractor to make money. So there is indeed a balance, of a kind. But it is not a deliberate balancing of two interests each with claim to special consideration. It is a balance between a public goal and market forces. The government tries to obtain for the taxpaying motorists the best deal they can get in the context of a free society and a free market.

In the copyright bargain, the government spends our freedom instead of our money. Freedom is more precious than money, so government's responsibility to spend our freedom wisely and frugally is even greater than its responsibility to spend our money thus.

than its responsibility to spend our money thus. Governments must never put the publishers' interests on a par with the public's freedom.

Not “Balance” but “Trade-Off”

The idea of balancing the readers' interests against the publishers' is the wrong way to judge copyright policy, but there are indeed two interests to be weighed: two interests *of the readers*. Readers have an interest in their own freedom in using published works; depending on circumstances, they may also have an interest in encouraging publication through some kind of incentive system.

The word “balance,” in discussions of copyright, has come to stand as shorthand for the idea of “striking a balance” between the readers and the publishers. Therefore, to use the word “balance” in regard to the readers' two interests would be confusing. We need another term.

In general, when one party has two goals that partly conflict, and cannot completely achieve both of them, we

call this a “trade-off.” Therefore, rather than speaking of “striking the right balance” between parties, we should speak of “finding the right trade-off between spending our freedom and keeping it.”

The Second Error: Maximizing One Output

The second mistake in copyright policy consists of adopting the goal of maximizing—not just increasing—the number of published works. The erroneous concept of “striking a balance” elevated the publishers to parity with the readers; this second error places them far above the readers.

When we purchase something, we do not generally buy the whole quantity in stock or the most expensive model. Instead we conserve funds for other purchases, by buying only what we need of any particular good, and choosing a model of sufficient rather than highest quality. The principle of diminishing returns suggests that spending all our money on one particular good is likely to be an inefficient allocation of resources; we generally choose to keep some money for another use.

Diminishing returns applies to copyright just as to any other purchase. The first freedoms we should trade away are those we miss the least, and whose sacrifice gives the largest encouragement to publication. As we trade additional freedoms that cut closer to home, we find that each trade is a bigger sacrifice than the last, while bringing a smaller increment in literary activity. Well before the increment becomes zero, we may well say it is not worth its incremental price; we would then settle on a bargain whose overall result is to increase the amount of publication, but not to the utmost possible extent.

Accepting the goal of maximizing publication rejects all these wiser, more advantageous bargains in advance—it dictates that the public must cede nearly all of its freedom to use published works, for just a little more publication.

The Rhetoric of Maximization

In practice, the goal of maximizing publication regardless of the cost to freedom is supported by widespread rhetoric which asserts that public copying is illegitimate

rhetoric which asserts that public copying is illegitimate, unfair, and intrinsically wrong. For instance, the publishers call people who copy “pirates,” a smear term designed to equate sharing information with your neighbor with attacking a ship. (This smear term was formerly used by authors to describe publishers who found lawful ways to publish unauthorized editions; its modern use by the publishers is almost the reverse.) This rhetoric directly rejects the constitutional basis for copyright, but presents itself as representing the unquestioned tradition of the American legal system.

The “pirate” rhetoric is typically accepted because it so pervades the media that few people realize how radical it is. It is effective because if copying by the public is fundamentally illegitimate, we can never object to the publishers’ demand that we surrender our freedom to do so. In other words, when the public is challenged to show why publishers should not receive some additional power, the most important reason of all—“We want to copy”—is disqualified in advance.

This leaves no way to argue against increasing copyright power except using side issues. Hence,

opposition to stronger copyright powers today almost exclusively cites side issues, and never dares cite the freedom to distribute copies as a legitimate public value.

As a practical matter, the goal of maximization enables publishers to argue that “A certain practice is reducing our sales—or we think it might—so we presume it diminishes publication by some unknown amount, and therefore it should be prohibited.” We are led to the outrageous conclusion that the public good is measured by publishers’ sales: What’s good for General Media is good for the USA.

The Third Error: Maximizing Publishers’ Power

Once the publishers have obtained assent to the policy goal of maximizing publication output at any cost, their next step is to infer that this requires giving them the maximum possible powers—making copyright cover every imaginable use of a work, or applying some other legal tool such as “shrink wrap” licenses to equivalent effect. This goal, which entails the abolition of “fair use”

and the “right of first sale,” is being pressed at every available level of government, from states of the US to international bodies.

This step is erroneous because strict copyright rules obstruct the creation of useful new works. For instance, Shakespeare borrowed the plots of some of his plays from works others had published a few decades before, so if today’s copyright law had been in effect, his plays would have been illegal.

Even if we wanted the highest possible rate of publication, regardless of cost to the public, maximizing publishers’ power is the wrong way to get it. As a means of promoting progress, it is self-defeating.

The Results of the Three Errors

The current trend in copyright legislation is to hand publishers broader powers for longer periods of time. The conceptual basis of copyright, as it emerges distorted from the series of errors, rarely offers a basis for saying no. Legislators give lip service to the idea that copyright serves the public while in fact giving publishers

copyright serves the public, while in fact giving publishers whatever they ask for.

For example, here is what Senator Hatch said when introducing S. 483, [\[2\]](#) a 1995 bill to increase the term of copyright by 20 years:

I believe we are now at such a point with respect to the question of whether the current term of copyright adequately protects the interests of authors and the related question of whether the term of protection continues to provide a sufficient incentive for the creation of new works of authorship. [\[3\]](#)

This bill extended the copyright on already published works written since the 1920s. This change was a giveaway to publishers with no possible benefit to the public, since there is no way to retroactively increase now the number of books published back then. Yet it cost the public a freedom that is meaningful today—the freedom to redistribute books from that era.

The bill also extended the copyrights of works yet to be written. For works made for hire, copyright would last 95 years instead of the present 75 years. Theoretically this would increase the incentive to write

theoretically this would increase the incentive to write new works; but any publisher that claims to need this extra incentive should be required to substantiate the claim with projected balance sheets for 75 years in the future.

Needless to say, Congress did not question the publishers' arguments: a law extending copyright was enacted in 1998. It was officially called the Sonny Bono Copyright Term Extension Act, named after one of its sponsors who died earlier that year. We usually call it the Mickey Mouse Copyright Act, since we presume its real motive was to prevent the copyright on the appearance of Mickey Mouse from expiring. Bono's widow, who served the rest of his term, made this statement:

Actually, Sonny wanted the term of copyright protection to last forever. I am informed by staff that such a change would violate the Constitution. I invite all of you to work with me to strengthen our copyright laws in all of the ways available to us. As you know, there is also Jack Valenti's [\[4\]](#) proposal for term to last forever less one day. Perhaps the Committee may look at that next Congress. [\[5\]](#)

The Supreme Court later heard a case that sought to

The Supreme Court later heard a case that sought to overturn the law on the grounds that the retroactive extension fails to serve the Constitution's goal of promoting progress. The court responded by abdicating its responsibility to judge the question; on copyright, the Constitution requires only lip service.

Another law, passed in 1997, made it a felony to make sufficiently many copies of any published work, even if you give them away to friends just to be nice. Previously this was not a crime in the US at all.

An even worse law, the Digital Millennium Copyright Act (DMCA), was designed to bring back copy protection (which computer users detest) by making it a crime to break copy protection, or even publish information about how to break it. This law ought to be called the "Domination by Media Corporations Act" because it effectively offers publishers the chance to write their own copyright law. It says they can impose any restrictions whatsoever on the use of a work, and these restrictions take the force of law provided the work contains some sort of encryption or license manager to enforce them.

One of the arguments offered for this bill was that it would implement a recent treaty to increase copyright powers. The treaty was promulgated by the World “Intellectual Property” Organization, an organization dominated by copyright- and patent-holding interests, with the aid of pressure from the Clinton administration; since the treaty only increases copyright power, whether it serves the public interest in any country is doubtful. In any case, the bill went far beyond what the treaty required.

Libraries were a key source of opposition to this bill, especially to the aspects that block the forms of copying that are considered fair use. How did the publishers respond? Former representative Pat Schroeder, now a lobbyist for the Association of American Publishers, said that the publishers “could not live with what [the libraries were] asking for.” Since the libraries were asking only to preserve part of the status quo, one might respond by wondering how the publishers had survived until the present day.

Congressman Barney Frank, in a meeting with me and others who opposed this bill, showed how far the US Constitution's view of copyright has been disregarded. He said that new powers, backed by criminal penalties, were needed urgently because the "movie industry is worried," as well as the "music industry" and other "industries." I asked him, "But is this in the public interest?" His response was telling: "Why are you talking about the public interest? These creative people don't have to give up their rights for the public interest!" The "industry" has been identified with the "creative people" it hires, copyright has been treated as its entitlement, and the Constitution has been turned upside down.

The DMCA was enacted in 1998. As enacted, it says that fair use remains nominally legitimate, but allows publishers to prohibit all software or hardware that you could practice it with. Effectively, fair use is prohibited.

Based on this law, the movie industry has imposed censorship on free software for reading and playing DVDs, and even on the information about how to read them. In April 2001, Professor Edward Felten of

Princeton University was intimidated by lawsuit threats from the Recording Industry Association of America (RIAA) into withdrawing a scientific paper stating what he had learned about a proposed encryption system for restricting access to recorded music.

We are also beginning to see e-books that take away many of readers' traditional freedoms—for instance, the freedom to lend a book to your friend, to sell it to a used book store, to borrow it from a library, to buy it without giving your name to a corporate data bank, even the freedom to read it twice. Encrypted e-books generally restrict all these activities—you can read them only with special secret software designed to restrict you.

I will never buy one of these encrypted, restricted e-books, and I hope you will reject them too. If an e-book doesn't give you the same freedoms as a traditional paper book, don't accept it!

Anyone independently releasing software that can read restricted e-books risks prosecution. A Russian programmer, Dmitry Sklyarov, was arrested in 2001 while visiting the US to speak at a conference because

while visiting the US to speak at a conference, because he had written such a program in Russia, where it was lawful to do so. Now Russia is preparing a law to prohibit it too, and the European Union recently adopted one.

Mass-market e-books have been a commercial failure so far, but not because readers chose to defend their freedom; they were unattractive for other reasons, such as that computer display screens are not easy surfaces to read from. We can't rely on this happy accident to protect us in the long term; the next attempt to promote e-books will use "electronic paper"—book-like objects into which an encrypted, restricted e-book can be downloaded. If this paper-like surface proves more appealing than today's display screens, we will have to defend our freedom in order to keep it. Meanwhile, e-books are making inroads in niches: NYU and other dental schools require students to buy their textbooks in the form of restricted e-books.

The media companies are not satisfied yet. In 2001, Disney-funded Senator Hollings proposed a bill called the "Security Systems Standards and Certification Act"

(SSSCA), [6] which would require all computers (and other digital recording and playback devices) to have government-mandated copy-restriction systems. That is their ultimate goal, but the first item on their agenda is to prohibit any equipment that can tune digital HDTV unless it is designed to be impossible for the public to “tamper with” (i.e., modify for their own purposes). Since free software is software that users can modify, we face here for the first time a proposed law that explicitly prohibits free software for a certain job. Prohibition of other jobs will surely follow. If the FCC adopts this rule, existing free software such as GNU Radio would be censored.

To block these bills and rules requires political action. [7]

Finding the Right Bargain

What is the proper way to decide copyright policy? If copyright is a bargain made on behalf of the public, it should serve the public interest above all. The government's duty when selling the public's freedom is to sell only what it must, and sell it as dearly as possible. At

the very least, we should pare back the extent of copyright as much as possible while maintaining a comparable level of publication.

Since we cannot find this minimum price in freedom through competitive bidding, as we do for construction projects, how can we find it?

One possible method is to reduce copyright privileges in stages, and observe the results. By seeing if and when measurable diminutions in publication occur, we will learn how much copyright power is really necessary to achieve the public's purposes. We must judge this by actual observation, not by what publishers say will happen, because they have every incentive to make exaggerated predictions of doom if their powers are reduced in any way.

Copyright policy includes several independent dimensions, which can be adjusted separately. After we find the necessary minimum for one policy dimension, it may still be possible to reduce other dimensions of copyright while maintaining the desired publication level.

One important dimension of copyright is its duration, which is now typically on the order of a century. Reducing the monopoly on copying to ten years, starting from the date when a work is published, would be a good first step. Another aspect of copyright, which covers the making of derivative works, could continue for a longer period.

Why count from the date of publication? Because copyright on unpublished works does not directly limit readers' freedom; whether we are free to copy a work is moot when we do not have copies. So giving authors a longer time to get a work published does no harm. Authors (who generally do own the copyright prior to publication) will rarely choose to delay publication just to push back the end of the copyright term.

Why ten years? Because that is a safe proposal; we can be confident on practical grounds that this reduction would have little impact on the overall viability of publishing today. In most media and genres, successful works are very profitable in just a few years, and even successful works are usually out of print well before ten.

Even for reference works, whose useful life may be many decades, ten-year copyright should suffice: updated editions are issued regularly, and many readers will buy the copyrighted current edition rather than copy a ten-year-old public domain version.

Ten years may still be longer than necessary; once things settle down, we could try a further reduction to tune the system. At a panel on copyright at a literary convention, where I proposed the ten-year term, a noted fantasy author sitting beside me objected vehemently, saying that anything beyond five years was intolerable.

But we don't have to apply the same time span to all kinds of works. Maintaining the utmost uniformity of copyright policy is not crucial to the public interest, and copyright law already has many exceptions for specific uses and media. It would be foolish to pay for every highway project at the rates necessary for the most difficult projects in the most expensive regions of the country; it is equally foolish to "pay" for all kinds of art with the greatest price in freedom that we find necessary for any one kind.

So perhaps novels, dictionaries, computer programs, songs, symphonies, and movies should have different durations of copyright, so that we can reduce the duration for each kind of work to what is necessary for many such works to be published. Perhaps movies over one hour long could have a 20-year copyright, because of the expense of producing them. In my own field, computer programming, three years should suffice, because product cycles are even shorter than that.

Another dimension of copyright policy is the extent of fair use: some ways of reproducing all or part of a published work that are legally permitted even though it is copyrighted. The natural first step in reducing this dimension of copyright power is to permit occasional private small-quantity noncommercial copying and distribution among individuals. This would eliminate the intrusion of the copyright police into people's private lives, but would probably have little effect on the sales of published works. (It may be necessary to take other legal steps to ensure that shrink-wrap licenses cannot be used to substitute for copyright in restricting such copying.) The experience of Napster shows that we should also

permit noncommercial verbatim redistribution to the general public—when so many of the public want to copy and share, and find it so useful, only draconian measures will stop them, and the public deserves to get what it wants.

For novels, and in general for works that are used for entertainment, noncommercial verbatim redistribution may be sufficient freedom for the readers. Computer programs, being used for functional purposes (to get jobs done), call for additional freedoms beyond that, including the freedom to publish an improved version. See “The Free Software Definition,” in this book, for an explanation of the freedoms that software users should have. But it may be an acceptable compromise for these freedoms to be universally available only after a delay of two or three years from the program’s publication.

Changes like these could bring copyright into line with the public’s wish to use digital technology to copy. Publishers will no doubt find these proposals “unbalanced”; they may threaten to take their marbles and go home, but they won’t really do it, because the game will remain profitable and it will be the only game in

game will remain profitable and it will be the only game in town.

As we consider reductions in copyright power, we must make sure media companies do not simply replace it with end-user license agreements. It would be necessary to prohibit the use of contracts to apply restrictions on copying that go beyond those of copyright. Such limitations on what mass-market nonnegotiated contracts can require are a standard part of the US legal system.

A Personal Note

I am a software designer, not a legal scholar. I've become concerned with copyright issues because there's no avoiding them in the world of computer networks, such as the Internet. As a user of computers and networks for 30 years, I value the freedoms that we have lost, and the ones we may lose next. As an author, I can reject the romantic mystique of the author as semidivine creator, often cited by publishers to justify increased copyright powers for authors—powers which these authors will then sign away to publishers.

Most of this article consists of facts and reasoning that you can check, and proposals on which you can form your own opinions. But I ask you to accept one thing on my word alone: that authors like me don't deserve special power over you. If you wish to reward me further for the software or books I have written, I would gratefully accept a check—but please don't surrender your freedom in my name.

Endnotes

1 *Fox Film Corp. v. Doyal*, 286 US 123, 1932.

2 *Congressional Record*, S. 483, “The Copyright Term Extension Act of 1995,” 2 March 1995, pp. S3390–4.

3 *Congressional Record*, “Statement on Introduced Bills and Joint Resolutions,” 2 March 1995, p. S3390,
<http://gpo.gov//fdsys//pkg//CREC-1995-03-02//pdf//CREC-1995-03-02-pt1-PgS3390-2.pdf>.

4 Jack Valenti was a longtime president of the Motion Picture Association of America.

5 *Congressional Record*, remarks of Rep. Bono, 7 October 1998, p. H9952, <http://gpo.gov/fdsys/pkg/CREC-1998-10-07/pdf//CREC-1998-10-07-pt1-PgH9946.pdf>.

6 Since renamed to the unpronounceable CBDTPA, for which a good mnemonic is “Consume, But Don’t Try Programming Anything” but it really stands for the “Consumer Broadband and Digital Television Promotion Act.”

7 If you would like to help, I recommend the web sites <http://defectivebydesign.org>, <http://publicknowledge.org>, and <http://eff.org>.

Chapter 19

Science Must Push Copyright Aside

Copyright © 2001 Richard Stallman

This essay was first published in *Nature* magazine's *Web Debates* forum, on 8 June 2001. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

It should be a truism that the scientific literature exists to disseminate scientific knowledge, and that scientific journals exist to facilitate the process. It therefore follows that rules for use of the scientific literature should be designed to help achieve that goal.

The rules we have now, known as copyright, were established in the age of the printing press, an inherently centralized method of mass-production copying. In a

print environment, copyright on journal articles restricted only journal publishers—requiring them to obtain permission to publish an article—and would-be plagiarists. It helped journals to operate and disseminate knowledge, without interfering with the useful work of scientists or students, either as writers or readers of articles. These rules fit that system well.

The modern technology for scientific publishing, however, is the World Wide Web. What rules would best ensure the maximum dissemination of scientific articles, and knowledge, on the web? Articles should be distributed in nonproprietary formats, with open access for all. And everyone should have the right to “mirror” articles—that is, to republish them verbatim with proper attribution.

These rules should apply to past as well as future articles, when they are distributed in electronic form. But there is no crucial need to change the present copyright system as it applies to paper publication of journals because the problem is not in that domain.

I Unfortunately, it seems that not everyone agrees with

Unfortunately, it seems that not everyone agrees with the truisms that began this article. Many journal publishers appear to believe that the purpose of scientific literature is to enable them to publish journals so as to collect subscriptions from scientists and students. Such thinking is known as “confusion of the means with the ends.”

Their approach has been to restrict access even to read the scientific literature to those who can and will pay for it. They use copyright law, which is still in force despite its inappropriateness for computer networks, as an excuse to stop scientists from choosing new rules.

For the sake of scientific cooperation and humanity's future, we must reject that approach at its root—not merely the obstructive systems that have been instituted, but the mistaken priorities that inspired them.

Journal publishers sometimes claim that online access requires expensive high-powered server machines, and that they must charge access fees to pay for these servers. This “problem” is a consequence of its own “solution.” Give everyone the freedom to mirror, and

libraries around the world will set up mirror sites to meet the demand. This decentralized solution will reduce network bandwidth needs and provide faster access, all the while protecting the scholarly record against accidental loss.

Publishers also argue that paying the editors requires charging for access. Let us accept the assumption that editors must be paid; this tail need not wag the dog. The cost of editing for a typical paper is between 1 percent and 3 percent of the cost of funding the research to produce it. Such a small percentage of the cost can hardly justify obstructing the use of the results.

Instead, the cost of editing could be recovered, for example, through page charges to the authors, who can pass these on to the research sponsors. The sponsors should not mind, given that they currently pay for publication in a more cumbersome way, through overhead fees for the university library's subscription to the journal. By changing the economic model to charge editing costs to the research sponsors, we can eliminate the apparent need to restrict access. The occasional author who is not affiliated with an institution or

author who is not affiliated with an institution or company, and who has no research sponsor, could be exempted from page charges, with costs levied on institution-based authors.

Another justification for access fees to online publications is to fund conversion of the print archives of a journal into online form. That work needs to be done, but we should seek alternative ways of funding it that do not involve obstructing access to the result. The work itself will not be any more difficult, or cost any more. It is self-defeating to digitize the archives and waste the results by restricting access.

The US Constitution says that copyright exists “to promote the Progress of Science.” When copyright impedes the progress of science, science must push copyright out of the way.

Chapter 20

Freedom—or Copyright

Copyright © 2008, 2010 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2008.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

This essay addresses how the principles of software freedom apply in some cases to other works of authorship and art. It's included here since it involves the application of the ideas of free software.

Copyright was established in the age of the printing press as an industrial regulation on the business of writing and publishing. The aim was to encourage the publication of a diversity of written works. The means was to require publishers to get the author's permission to publish recent writings. This enabled authors to get income from publishers, which facilitated and encouraged writing. The

general reading public received the benefit of this, while losing little: copyright restricted only publication, not the things an ordinary reader could do. That made copyright arguably a beneficial system for the public, and therefore arguably legitimate.

Well and good—back then.

Now we have a new way of distributing information: computers and networks. Their benefit is that they facilitate copying and manipulating information, including software, musical recordings, books, and movies. They offer the possibility of unlimited access to all sorts of data—an information utopia.

One obstacle stood in the way: copyright. Readers and listeners who made use of their new ability to copy and share published information were technically copyright infringers. The same law which had formerly acted as a beneficial industrial regulation on publishers had become a restriction on the public it was meant to serve.

In a democracy, a law that prohibits a popular and useful activity is usually soon relaxed. Not so where corporations have political power. The publishers' lobby

was determined to prevent the public from taking advantage of the power of their computers, and found copyright a handy weapon. Under their influence, rather than relaxing copyright rules to suit the new circumstances, governments made them stricter than ever, imposing harsh penalties on the practice of sharing. The latest fashion in supporting the publishers against the citizens, known as “three strikes,” is to cut off people’s Internet connections if they share.

But that wasn’t the worst of it. Computers can be powerful tools of domination when software suppliers deny users the control of the software they run. The publishers realized that by publishing works in encrypted format, which only specially authorized software could view, they could gain unprecedented power: they could compel readers to pay, and identify themselves, every time they read a book, listen to a song, or watch a video. That is the publishers’ dream: a pay-per-view universe.

The publishers gained US government support for their dream with the Digital Millennium Copyright Act of 1998. This law gave publishers power to write their own copyright rules, by implementing them in the code of the authorized player software. Under this practice, called Digital Restrictions Management, or DRM, even reading

Digital Restrictions Management, or DRM, even reading or listening without authorization is forbidden.

We still have the same old freedoms in using paper books and other analog media. But if e-books replace printed books, those freedoms will not transfer. Imagine: no more used book stores; no more lending a book to your friend; no more borrowing one from the public library—no more “leaks” that might give someone a chance to read without paying. No more purchasing a book anonymously with cash—you can only buy an e-book with a credit card. That is the world the publishers want to impose on us. If you buy the Amazon Kindle (we call it the Swindle) or the Sony Reader (we call it the Shreader for what it threatens to do to books), you pay to establish that world.

The Swindle even has an Orwellian back door that can be used to erase books remotely. Amazon demonstrated this capability by erasing copies, purchased from Amazon, of Orwell’s book *1984*. Evidently Amazon’s name for this product reflects the intention to burn our books.

Public anger against DRM is slowly growing, held back because propaganda expressions such as “protect

authors” and “intellectual property” have convinced readers that their rights do not count. These terms implicitly assume that publishers deserve special power in the name of the authors, that we are morally obliged to bow to them, and that we have wronged someone if we see or hear anything without paying for permission.

The organizations that profit most from copyright legally exercise it in the name of the authors (most of whom gain little). They would have you believe that copyright is a natural right of authors, and that we the public must suffer it no matter how painful it is. They call sharing “piracy,” equating helping your neighbor with attacking a ship.

They also tell us that a War on Sharing is the only way to keep art alive. Even if true, it would not justify the policy; but it isn’t true. Public sharing of copies is likely to increase the sales of most works, and decrease sales only for big hits.

Bestsellers can still do well without forbidding sharing. Stephen King got hundreds of thousands of dollars selling an unencrypted e-book serial with no obstacle to copying and sharing. (He was dissatisfied with that amount and called the experiment a failure, but

with that amount and called the experiment a failure, but it looks like a success to me.) Radiohead made millions in 2007 by inviting fans to copy an album and pay what they wished, while it was also shared through peer-to-peer. In 2008, Nine Inch Nails released an album with permission to share copies and made \$750,000 in a few days. [\[1\]](#)

The possibility of success without oppression is not limited to bestsellers. Many artists of various levels of fame now make an adequate living through voluntary support: [\[2\]](#) donations and merchandise purchases of their fans. Kevin Kelly [\[3\]](#) estimates the artist need only find around 1,000 true fans. [\[4\]](#)

When computer networks provide an easy anonymous method for sending someone a small amount of money, without a credit card, it will be easy to set up a much better system to support the arts. When you view a work, there will be a button you can press saying, “Click here to send the artist one dollar.” Wouldn’t you press it, at least once a week?

Another good way to support music and the arts is with tax funds—perhaps a tax on blank media or on Internet connectivity. The state should distribute the tax

money entirely to the artists, not waste it on corporate executives. But the state should not distribute it in linear proportion to popularity, because that would give most of it to a few superstars, leaving little to support all the other artists. I therefore recommend using a cube-root function or something similar. With linear proportion, superstar A with 1,000 times the popularity of a successful artist B will get 1,000 times as much money as B. With the cube root, A will get 10 times as much as B. Thus, each superstar gets a larger share than a less popular artist, but most of the funds go to the artists who really need this support. This system will use our tax money efficiently to support the arts.

The Global Patronage [\[5\]](#) proposal combines aspects of those two systems, incorporating mandatory payments with voluntary allocation among artists.

In Spain, this tax system should replace the SGAE [\[6\]](#) and its canon, which could be eliminated.

To make copyright fit the network age, we should legalize the noncommercial copying and sharing of all published works, and prohibit DRM. But until we win this battle, you must protect yourself: don't buy any products with DRM unless you personally have the

product that you are using, you are not allowed to use it in a way that means to break the DRM. Never use a product designed to attack your freedom unless you can nullify the attack.

Endnotes

1 “Nine Inch Nails Made at Least \$750k from CC Release in Two Days,” posted by Cory Doctorow, 5 March 2008, <http://boingboing.net/2008/03/05/nine-inch-nails-made.html>.

2 Mike Masnick, “The Future of Music Business Models (and Those Who Are Already There),” 25 January 2010, <http://techdirt.com/articles/20091119/1634117011.shtml>.

3 Kevin Kelly is a commentator on digital culture and the founder of *Wired* magazine.

4 Kevin Kelly, “1,000 True Fans,” 4 March 2008, http://kk.org/thetechnium/archives/2008/03/1000_true_fans.php.

5 See <http://mecenatglobal.org/> for more information.

6 The SGAE is Spain’s main copyright collective for composers, authors, and publishers.

Chapter 21

What Is Copyleft?

Copyright © 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009 Free Software Foundation, Inc.

This essay was originally published on <http://gnu.org>, in 1996.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Copyleft is a general method for making a program (or other work) free, and requiring all modified and extended versions of the program to be free as well.

The simplest way to make a program free software is to put it in the public domain, uncopyrighted. This allows people to share the program and their improvements, if they are so minded. But it also allows uncooperative people to convert the program into proprietary software.

People can convert the program into proprietary software. They can make changes, many or few, and distribute the result as a proprietary product. People who receive the program in that modified form do not have the freedom that the original author gave them; the middleman has stripped it away.

In the GNU Project, our aim is to give *all* users the freedom to redistribute and change GNU software. If middlemen could strip off the freedom, we might have many users, but those users would not have freedom. So instead of putting GNU software in the public domain, we “copyleft” it. Copyleft says that anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it. Copyleft guarantees that every user has freedom.

Copyleft also provides an incentive for other programmers to add to free software. Important free programs such as the GNU C++ compiler exist only because of this.

Copyleft also helps programmers who want to contribute improvements to free software get permission

to do so. These programmers often work for companies or universities that would do almost anything to get more money. A programmer may want to contribute her changes to the community, but her employer may want to turn the changes into a proprietary software product.

When we explain to the employer that it is illegal to distribute the improved version except as free software, the employer usually decides to release it as free software rather than throw it away.

To copyleft a program, we first state that it is copyrighted; then we add distribution terms, which are a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code, *or any program derived from it*, but only if the distribution terms are unchanged. Thus, the code and the freedoms become legally inseparable.

Proprietary software developers use copyright to take away the users' freedom; we use copyright to guarantee their freedom. That's why we reverse the name, changing "copyright" into "copyleft."

Copyleft is a way of using of the copyright on the program. It doesn't mean abandoning the copyright; in fact, doing so would make copyleft impossible. The "left" in "copyleft" is not a reference to the verb "to leave"—only to the direction which is the inverse of "right."

Copyleft is a general concept, and you can't use a general concept directly; you can only use a specific implementation of the concept. In the GNU Project, the specific distribution terms that we use for most software are contained in the GNU General Public License ([265](#)). The GNU General Public License is often called the GNU GPL for short. There is also a Frequently Asked Questions page about the GNU GPL, at <http://gnu.org/licenses/gpl-faq.html>. You can also read about why the FSF gets copyright assignments from contributors, at <http://gnu.org/copyleft/why-assign.html>.

An alternate form of copyleft, the GNU Lesser General Public License (LGPL) ([289](#)), applies to a few (but not all) GNU libraries. To learn more about properly using the LGPL, please read the article "Why You Shouldn't Use the Lesser GPL for Your Next Library."

shouldn't use the Lesser GPL for your next library, available at <http://gnu.org/philosophy/why-not-lgpl.html>.

The GNU Free Documentation License (FDL) ([293](#)) is a form of copyleft intended for use on a manual, textbook or other document to assure everyone the effective freedom to copy and redistribute it, with or without modifications, either commercially or noncommercially.

The appropriate license is included in many manuals and in each GNU source code distribution.

All these licenses are designed so that you can easily apply them to your own works, assuming you are the copyright holder. You don't have to modify the license to do this, just include a copy of the license in the work, and add notices in the source files that refer properly to the license.

Using the same distribution terms for many different programs makes it easy to copy code between various different programs. When they all have the same

distribution terms, there is no problem. The Lesser GPL, version 2, includes a provision that lets you alter the distribution terms to the ordinary GPL, so that you can copy code into another program covered by the GPL. Version 3 of the Lesser GPL is built as an exception added to GPL version 3, making the compatibility automatic.

If you would like to copyleft your program with the GNU GPL or the GNU LGPL, please see the license instructions page, at <http://gnu.org/copyleft/gpl-howto.html>, for advice. Please note that you must use the entire text of the license you choose. Each is an integral whole, and partial copies are not permitted.

If you would like to copyleft your manual with the GNU FDL, please see the instructions at the end of the FDL text ([302](#)), and the GFDL instructions page, at <http://gnu.org/copyleft/fdl-howto.html>. Again, partial copies are not permitted.

Chapter 22

Copyleft: Pragmatic Idealism

Copyright © 1998, 2003 Free Software Foundation, Inc.

This version of this essay is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Every decision a person makes stems from the person's values and goals. People can have many different goals and values; fame, profit, love, survival, fun, and freedom, are just some of the goals that a good person might have. When the goal is a matter of principle, we call that idealism.

My work on free software is motivated by an idealistic goal: spreading freedom and cooperation. I want to encourage free software to spread, replacing

proprietary software that forbids cooperation, and thus make our society better.

That's the basic reason why the GNU General Public License is written the way it is—as a copyleft. All code added to a GPL-covered program must be free software, even if it is put in a separate file. I make my code available for use in free software, and not for use in proprietary software, in order to encourage other people who write software to make it free as well. I figure that since proprietary software developers use copyright to stop us from sharing, we cooperators can use copyright to give other cooperators an advantage of their own: they can use our code.

Not everyone who uses the GNU GPL has this goal. Many years ago, a friend of mine was asked to rerelease a copylefted program under noncopyleft terms, and he responded more or less like this: “Sometimes I work on free software, and sometimes I work on proprietary software—but when I work on proprietary software, I expect to get *paid*.”

He was willing to share his work with a community

He was willing to share his work with a community that shares software, but saw no reason to give a handout to a business making products that would be off-limits to our community. His goal was different from mine, but he decided that the GNU GPL was useful for his goal too.

If you want to accomplish something in the world, idealism is not enough—you need to choose a method that works to achieve the goal. In other words, you need to be “pragmatic.” Is the GPL pragmatic? Let’s look at its results.

Consider GNU C++. Why do we have a free C++ compiler? Only because the GNU GPL said it had to be free. GNU C++ was developed by an industry consortium, MCC, starting from the GNU C compiler. MCC normally makes its work as proprietary as can be. But they made the C++ front end free software, because the GNU GPL said that was the only way they could release it. The C++ front end included many new files, but since they were meant to be linked with GCC, the GPL did apply to them. The benefit to our community is evident.

Consider GNU Objective C. NeXT initially wanted to make this front end proprietary; they proposed to release it as .o files, and let users link them with the rest of GCC, thinking this might be a way around the GPL's requirements. But our lawyer said that this would not evade the requirements, that it was not allowed. And so they made the Objective C front end free software.

Those examples happened years ago, but the GNU GPL continues to bring us more free software.

Many GNU libraries are covered by the GNU Lesser General Public License, but not all. One GNU library which is covered by the ordinary GNU GPL is Readline, which implements command-line editing. I once found out about a nonfree program which was designed to use Readline, and told the developer this was not allowed. He could have taken command-line editing out of the program, but what he actually did was rerelease it under the GPL. Now it is free software.

The programmers who write improvements to GCC (or Emacs. or Bash. or Linux. or any GPL-covered

(or others, or doing or having or any other covered program) are often employed by companies or universities. When the programmer wants to return his improvements to the community, and see his code in the next release, the boss may say, “Hold on there—your code belongs to us! We don’t want to share it; we have decided to turn your improved version into a proprietary software product.”

Here the GNU GPL comes to the rescue. The programmer shows the boss that this proprietary software product would be copyright infringement, and the boss realizes that he has only two choices: release the new code as free software, or not at all. Almost always he lets the programmer do as he intended all along, and the code goes into the next release.

The GNU GPL is not Mr. Nice Guy. It says no to some of the things that people sometimes want to do. There are users who say that this is a bad thing—that the GPL “excludes” some proprietary software developers who “need to be brought into the free software community.”

But we are not excluding them from our community; they are choosing not to enter. Their decision to make software proprietary is a decision to stay out of our community. Being in our community means joining in cooperation with us; we cannot “bring them into our community” if they don’t want to join.

What we *can* do is offer them an inducement to join. The GNU GPL is designed to make an inducement from our existing software: “If you will make your software free, you can use this code.” Of course, it won’t win ’em all, but it wins some of the time.

Proprietary software development does not contribute to our community, but its developers often want handouts from us. Free software users can offer free software developers strokes for the ego—recognition and gratitude—but it can be very tempting when a business tells you, “Just let us put your package in our proprietary program, and your program will be used by many thousands of people!” The temptation can be powerful, but in the long run we are all better off if we resist it.

The temptation and pressure are harder to recognize when they come indirectly, through free software organizations that have adopted a policy of catering to proprietary software. The X Consortium (and its successor, the Open Group) offers an example: funded by companies that made proprietary software, they strived for a decade to persuade programmers not to use copyleft. When the Open Group tried to make X11R6.4 nonfree software, those of us who had resisted that pressure were glad that we did.

In September 1998, several months after X11R6.4 was released with nonfree distribution terms, the Open Group reversed its decision and rereleased it under the same noncopyleft free software license that was used for X11R6.3. Thank you, Open Group—but this subsequent reversal does not invalidate the conclusions we draw from the fact that adding the restrictions was *possible*.

Pragmatically speaking, thinking about greater long-term goals will strengthen your will to resist this pressure. If you focus your mind on the freedom and community that you can build by staying firm, you will find the

strength to do it. "Stand for something, or you will fall for anything."

And if cynics ridicule freedom, ridicule community... if "hard-nosed realists" say that profit is the only ideal... just ignore them, and use copyleft all the same.

Part IV

**Software Patents: Danger to
Programmers**

Chapter 23

Anatomy of a Trivial Patent

Copyright © 2006 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2006.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Programmers are well aware that many of the existing software patents cover laughably obvious ideas. Yet the patent system's defenders often argue that these ideas are nontrivial, obvious only in hindsight. And it is surprisingly difficult to defeat them in debate. Why is that?

One reason is that any idea can be made to look complex when analyzed to death. Another reason is that these trivial ideas often look quite complex as described

in the patents themselves. The patent system's defenders can point to the complex description and say, "How can anything this complex be obvious?"

I will use an example to show you how. Here's claim number one from US patent number 5,963,916, applied for in October 1996:

1. A method for enabling a remote user to preview a portion of a pre-recorded music product from a network web site containing pre-selected portions of different pre-recorded music products, using a computer, a computer display and a telecommunications link between the remote user's computer and the network web site, the method comprising the steps of:

- using the remote user's computer to establish a telecommunications link to the network web site wherein the network web site comprises (i) a central host server coupled to a communications network for retrieving and transmitting the pre-selected portion of the pre-recorded music product upon request by a remote user and (ii) a central storage device for storing pre-selected portions of a plurality of different pre-recorded music products;
- transmitting user identification data from the remote user's

computer to the central host server thereby allowing the central host server to identify and track the user's progress through the network web site;

- choosing at least one pre-selected portion of the pre-recorded music products from the central host server;
- receiving the chosen pre-selected portion of the pre-recorded products; and
- interactively previewing the received chosen pre-selected portion of the pre-recorded music product.

That sure looks like a complex system, right? Surely it took a real clever guy to think of this? No, but it took cleverness to make it seem so complex. Let's analyze where the complexity comes from:

1. A method for enabling a remote user to preview a portion of a pre-recorded music product from a network web site containing pre-selected portions

That states the principal part of their idea. They put selections from certain pieces of music on a server so a user can listen to them.

of different pre-recorded music products,

This emphasizes their server stores selections from more than one piece of music.

It is a basic principle of computer science is that if a computer can do a thing once, it can do that thing many times, on different data each time. Many patents pretend that applying this principle to a specific case makes an “invention.”

using a computer, a computer display and a telecommunications link between the remote user's computer and the network web site,

This says they are using a server on a network.

the method comprising the steps of:

- using the remote user's computer to establish a telecommunications link to the network web site

This says that the user connects to the server over the network. (That's the way one uses a server.)

wherein the network web site comprises (i) a central host server coupled to a communications network

server coupled to a communications network

This informs us that the server is on the net. (That is typical of servers.)

for re-trieving and transmitting the pre-selected portion of the pre-recorded music product upon request by a remote user

This repeats the general idea stated in the first two lines.

and (ii) a central stor- age device for storing pre-selected portions of a plurality of different pre-recorded music products;

They have decided to put a hard disk (or equivalent) in their computer and store the music samples on that. Ever since around 1980, this has been the normal way to store anything on a computer for rapid access.

Note how they emphasize once again the fact that they can store more than one selection on this disk. Of course, every file system will let you store more than one file.

- transmitting user identification data from the remote user's computer to the central host server thereby allowing the central host server to identify and track the user's progress

central host server to identify and track the user's progress through the network web site;

This says that they keep track of who you are and what you access—a common (though nasty) thing for web servers to do. I believe it was common already in 1996.

- choosing at least one pre-selected portion of the pre-recorded music products from the central host server;

In other words, the user clicks to say which link to follow. That is typical for web servers; if they had found another way to do it, that might have been an invention.

- receiving the chosen pre-selected portion of the pre-recorded products; and

When you follow a link, your browser reads the contents. This is typical behavior for a web browser.

- interactively previewing the received chosen pre-selected portion of the pre-recorded music product.

This says that your browser plays the music for you. (That is what many browsers do, when you follow a link

(claim 1 is really an invention, they need to, then you have to add it to an audio file.)

Now you see how they padded this claim to make it into a complex idea: they combined their own idea (stated in two lines of text) with important aspects of what computers, networks, web servers, and web browsers do. This adds up to the so-called invention for which they received the patent.

This example is typical of software patents. Even the occasional patent whose idea is nontrivial has the same sort of added complication.

Now look at a subsequent claim:

3. The method of [149]claim 1 wherein the central memory device comprises a plurality of compact disc-read only memory (CD-ROMs).

What they are saying here is, “Even if you don’t think that claim 1 is really an invention, using CD-ROMs to store the data makes it an invention for sure. An average system designer would never have thought of storing data on a CD.”

Now look at the next claim:

4. The method of [150]claim 1 wherein the central memory device comprises a RAID array drive.

A RAID array is a group of disks set up to work like one big disk, with the special feature that, even if one of the disks in the array has a failure and stops working, all the data are still available on the other disks in the group. Such arrays have been commercially available since long before 1996, and are a standard way of storing data for high availability. But these brilliant inventors have patented the use of a RAID array for this particular purpose.

Trivial as it is, this patent would not necessarily be found legally invalid if there is a lawsuit about it. Not only the US Patent Office but the courts as well tend to apply a very low standard when judging whether a patent is “unobvious.” This patent might pass muster, according to them.

What’s more, the courts are reluctant to overrule the

Patent Office, so there is a better chance of getting a patent overturned if you can show a court prior art that the Patent Office did not consider. If the courts are willing to entertain a higher standard in judging unobviousness, it helps to save the prior art for them. Thus, the proposals to “make the system work better” by providing the Patent Office with a better database of prior art could instead make things worse.

It is very hard to make a patent system behave reasonably; it is a complex bureaucracy and tends to follow its structural imperatives regardless of what it is “supposed” to do. The only practical way to get rid of the many obvious patents on software features and business practices is to get rid of all patents in those fields. Fortunately, that would be no loss: the unobvious patents in the software field do no good either. What software patents do is put software developers and users under threat.

The patent system is supposed, intended, to promote progress, and those who benefit from software patents ask us to believe without question that they do have that effect. But programmers’ experience shows otherwise.

New theoretical analysis shows that this is no paradox.
(See

<http://researchoninnovation.org/patent.pdf>.)

There is no reason why society should expose software developers and users to the danger of software patents.

Chapter 24

Software Patents and Literary Patents

Copyright © 2005, 2007, 2008, 2009 Richard Stallman

This essay was originally published on <http://guardian.co.uk>, on 23 June 2005. It was then titled “Patent Absurdity” and focused on the proposed European software patent directive. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

When politicians consider the question of software patents, they are usually voting blind; not being programmers, they don't understand what software patents really do. They often think patents are similar to copyright law (“except for some details”)—which is not the case. For instance, when I publicly asked Patrick Devedjian, then Minister for Industry in France, how France would vote on the issue of software patents,

Devedjian responded with an impassioned defense of copyright law, praising Victor Hugo for his role in the adoption of copyright. (The misleading term “intellectual property” promotes this confusion—one of the reasons it should never be used.)

Those who imagine effects like those of copyright law cannot grasp the disastrous effects of software patents. We can use Victor Hugo as an example to illustrate the difference.

A novel and a modern complex program have certain points in common: each one is large, and implements many ideas in combination. So let's follow the analogy, and suppose that patent law had been applied to novels in the 1800s; suppose that states such as France had permitted the patenting of literary ideas. How would this have affected Victor Hugo's writing? How would the effects of literary patents compare with the effects of literary copyright?

Consider Victor Hugo's novel *Les Misérables*. Since he wrote it, the copyright belonged only to him. He did not have to fear that some stranger could sue him for

did not have to fear that some stranger could sue him for copyright infringement and win. That was impossible, because copyright covers only the details of a work of authorship, not the ideas embodied in them, and it only restricts copying. Hugo had not copied *Les Misérables*, so he was not in danger from copyright.

Patents work differently. Patents cover ideas; each patent is a monopoly on practicing some idea, which is described in the patent itself. Here's one example of a hypothetical literary patent:

- Claim 1: a communication process that represents in the mind of a reader the concept of a character who has been in jail for a long time and becomes bitter towards society and humankind.
- Claim 2: a communication process according to claim 1, wherein said character subsequently finds moral redemption through the kindness of another.
- Claim 3: a communication process according to claims 1 and 2, wherein said character changes his name during the story.

If such a patent had existed in 1862 when *Les*

Misérables was published, the novel would have conflicted with all three claims, since all these things happened to Jean Valjean in the novel. Victor Hugo could have been sued, and if sued, he would have lost. The novel could have been prohibited—in effect, censored—by the patent holder.

Now consider this hypothetical literary patent:

- Claim 1: a communication process that represents in the mind of a reader the concept of a character who has been in jail for a long time and subsequently changes his name.

Les Misérables would have been prohibited by that patent too, because this description too fits the life story of Jean Valjean. And here's another hypothetical patent:

- Claim 1: a communication process that represents in the mind of a reader the concept of a character who finds moral redemption and then changes his name.

Jean Valjean would have been forbidden by this patent

too.

All three patents would cover, and prohibit, the life story of this one character. They overlap, but they do not precisely duplicate each other, so they could all be valid simultaneously; all three patent holders could have sued Victor Hugo. Any one of them could have prohibited publication of *Les Misérables*.

This patent also could have been violated:

- Claim 1: a communication process that presents a character whose given name matches the last syllable of his family name.

through the name “Jean Valjean,” but at least this patent would have been easy to avoid.

You might think that these ideas are so simple that no patent office would have issued them. We programmers are often amazed by the simplicity of the ideas that real software patents cover—for instance, the European Patent Office has issued a patent on the progress bar, and a patent on accepting payment via credit cards.

These patents would be laughable if they were not so dangerous.

Other aspects of *Les Misérables* could also have run afoul of patents. For instance, there could have been a patent on a fictionalized portrayal of the Battle of Waterloo, or a patent on using Parisian slang in fiction. Two more lawsuits. In fact, there is no limit to the number of different patents that might have been applicable for suing the author of a work such as *Les Misérables*. All the patent holders would say they deserved a reward for the literary progress that their patented ideas represent, but these obstacles would not promote progress in literature, they would only obstruct it.

However, a very broad patent could have made all these issues irrelevant. Imagine a patent with broad claims like these:

- A communication process structured with narration that continues through many pages.
- A narration structure sometimes resembling a

logue or improvisation.

- Intrigue articulated around the confrontation of specific characters, each in turn setting traps for the others.
- Narration that presents many layers of society.
- Narration that shows the wheels of hidden conspiracy.

Who would the patent holders have been? They could have been other novelists, perhaps Dumas or Balzac, who had written such novels—but not necessarily. It isn't required to write a program to patent a software idea, so if our hypothetical literary patents follow the real patent system, these patent holders would not have had to write novels, or stories, or anything—except patent applications. Patent parasite companies, businesses that produce nothing except threats and lawsuits, are booming nowadays.

Given these broad patents, Victor Hugo would not have reached the point of asking what patents might get him sued for using the character of Jean Valjean, because he could not even have considered writing a novel of this kind.

This analogy can help nonprogrammers see what software patents do. Software patents cover features, such as defining abbreviations in a word processor, or natural order recalculation in a spreadsheet. Patents cover algorithms that programs need to use. Patents cover aspects of file formats, such as Microsoft's OOXML format. MPEG 2 video format is covered by 39 different US patents.

Just as one novel could run afoul of many different literary patents at once, one program can be prohibited by many different patents at once. It is so much work to identify all the patents that appear to apply to a large program that only one such study has been done. A 2004 study of Linux, the kernel of the GNU/Linux operating system, found 283 different US software patents that seemed to cover it. That is to say, each of these 283 different patents forbids some computational process found somewhere in the thousands of pages of source code of Linux. At the time, Linux was around 1 percent of the whole GNU/Linux system. How many patents might there be that a distributor of the whole

system could be sued under?

The way to prevent software patents from bollixing software development is simple: don't authorize them. This ought to be easy, since most patent laws have provisions against software patents. They typically say that "software per se" cannot be patented. But patent offices around the world are trying to twist the words and issuing patents on the ideas implemented in programs. Unless this is blocked, the result will be to put all software developers in danger.

Chapter 25

The Danger of Software Patents

Copyright © 2009 Richard Stallman

This transcript was originally published on <http://gnu.org>, in 2009.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

This chapter is licensed under the Creative Commons Attribution-NoDerivs 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

This is an unedited transcript of the talk presented by Richard Stallman on 8 October 2009 at Victoria University of Wellington, in Wellington, New Zealand.

I'm most known for starting the free software movement and leading development of the GNU operating system—although most of the people who use the system mistakenly believe it's Linux and think it was

started by somebody else a decade later. But I'm not going to be speaking about any of that today. I'm here to talk about a legal danger to all software developers, distributors, and users: the danger of patents—on computational ideas, computational techniques, an idea for something you can do on a computer.

Now, to understand this issue, the first thing you need to realize is that patent law has nothing to do with copyright law—they're totally different. Whatever you learn about one of them, you can be sure it doesn't apply to the other.

So, for example, any time a person makes a statement about “intellectual property,” that's spreading confusion, because it's lumping together not only these two laws but also at least a dozen others. They're all different, and the result is any statement which purports to be about “intellectual property” is pure confusion—either the person making the statement is confused, or the person is trying to confuse others. But either way, whether it's accidental or malicious, it's confusion.

Protect yourself from this confusion by rejecting any

Protect yourself from this confusion by rejecting any statement which makes use of that term. The only way to make thoughtful comments and think clear thoughts about any one of these laws is to distinguish it first from all the others, and talk or think about one particular law, so that we can understand what it actually does and then form conclusions about it. So I'll be talking about patent law, and what happens in those countries which have allowed patent law to restrict software.

So, what does a patent do? A patent is an explicit, government-issued monopoly on using a certain idea. In the patent there's a part called the claims, which describe exactly what you're not allowed to do (although they're written in a way you probably can't understand). It's a struggle to figure out what those prohibitions actually mean, and they may go on for many pages of fine print.

So the patent typically lasts for 20 years, which is a fairly long time in our field. Twenty years ago there was no World Wide Web—a tremendous amount of the use of computers goes on in an area which wasn't even possible to propose 20 years ago. So of course everything that people do on it is something that's new

since 20 years ago—at least in some aspect it is new. So if patents had been applied for we'd be prohibited from doing all of it, and we may be prohibited from doing all of it in countries that have been foolish enough to have such a policy.

Most of the time, when people describe the function of the patent system, they have a vested interest in the system. They may be patent lawyers, or they may work in the Patent Office, or they may be in the patent office of a megacorporation, so they want you to like the system.

The *Economist* once referred to the patent system as “a time-consuming lottery.” If you've ever seen publicity for a lottery, you understand how it works: they dwell on the very unlikely probability of winning, and they don't talk about the overwhelming likelihood of losing. In this way, they intentionally and systematically present a biased picture of what's likely to happen to you, without actually lying about any particular fact.

It's the same way for the publicity for the patent system: they talk about what it's like to walk down the street with a patent in your pocket—or first of all, what

street with a patent in your pocket—of first of all, what it's like to get a patent, then what it's like to have a patent in your pocket, and every so often you can pull it out and point it at somebody and say, "Give me your money."

To compensate for their bias, I'm going to describe it from the other side, the victim side—what it's like for people who want to develop or distribute or run software. You have to worry that any day someone might walk up to you and point a patent at you and say, "Give me your money."

If you want to develop software in a country that allows software patents, and you want to work with patent law, what will you have to do?

You could try to make a list of all the ideas that one might be able to find in the program that you're about to write, aside from the fact that you don't know that when you start writing the program. [But] even after you finish writing the program you wouldn't be able to make such a list.

The reason is...in the process you conceived of it in one particular way—you've got a mental structure to apply to your design. And because of that, it will block you from seeing other structures that somebody might use to understand the same program—because you're not coming to it fresh; you already designed it with one structure in mind. Someone else who sees it for the first time might see a different structure, which involves different ideas, and it would be hard for you to see what those other ideas are. But nonetheless they're implemented in your program, and those patents could prohibit your program, if those ideas are patented.

For instance, suppose there were graphical-idea patents and you wanted to draw a square. Well, you would realize that if there was a patent on a bottom edge, it would prohibit your square. You could put "bottom edge" on the list of all ideas implemented in your drawing. But you might not realize that somebody else with a patent on bottom corners could sue you easily also, because he could take your drawing and turn it by 45 degrees. And now your square is like this, and it has a bottom corner.

So you couldn't make a list of all the ideas which, if patented, could prohibit your program.

What you might try to do is find out all the ideas that are patented that might be in your program. Now you can't do that actually, because patent applications are kept secret for at least 18 months; and the result is the Patent Office could be considering now whether to issue a patent, and they won't tell you. And this is not just an academic, theoretical possibility.

For instance, in 1984 the Compress program was written, a program for compressing files using the data compression algorithm, and at that time there was no patent on that algorithm for compressing files. The author got the algorithm from an article in a journal. That was when we thought that the purpose of computer science journals was to publish algorithms so people could use them.

He wrote this program, he released it, and in 1985 a patent was issued on that algorithm. But the patent holder was cunning and didn't immediately go around telling

people to stop using it. The patent holder figured, "Let's let everybody dig their grave deeper." A few years later they started threatening people; it became clear we couldn't use Compress, so I asked for people to suggest other algorithms we could use for compressing files.

And somebody wrote and said, "I developed another data compression algorithm that works better, I've written a program, I'd like to give it to you." So we got ready to release it, and a week before it was ready to be released, I read in the *New York Times* weekly patent column, which I rarely saw—it's a couple of times a year I might see it—but just by luck I saw that someone had gotten a patent for "inventing a new method of compressing data." And so I said we had better look at this, and sure enough it covered the program we were about to release. But it could have been worse: the patent could have been issued a year later, or two years later, or three years later, or five years later.

Anyway, someone else came up with another, even better compression algorithm, which was used in the program *gzip*, and just about everybody who wanted to

program, and just used a program that would compress files switched to gzip, so it sounds like a happy ending. But you'll hear more later. It's not entirely so happy.

So, you can't find out about the patents that are being considered even though they may prohibit your work once they come out, but you can find out about the already issued patents. They're all published by the Patent Office. The problem is you can't read them all, because there are too many of them.

In the US I believe there are hundreds of thousands of software patents; keeping track of them would be a tremendous job. So you're going to have to search for relevant patents. And you'll find a lot of relevant patents, but you won't necessarily find them all.

For instance, in the 80s and 90s, there was a patent on "natural order recalculation" in spreadsheets. Somebody once asked me for a copy of it, so I looked in our computer file which lists the patent numbers. And then I pulled out the drawer to get the paper copy of this patent and xeroxed it and sent it to him. And when he got

it, he said, "I think you sent me the wrong patent. This is something about compilers." So I thought maybe our file has the wrong number in it. I looked in it again, and sure enough it said, "A method for compiling formulas into object code." So I started to read it to see if it was indeed the wrong patent. I read the claims, and sure enough it was the natural order recalculation patent, but it didn't use those terms. It didn't use the term "spreadsheet." In fact, what the patent prohibited was dozens of different ways of implementing topological sort—all the ways they could think of. But I don't think it used the term "topological sort."

So if you were writing a spreadsheet and you tried to find relevant patents by searching, you might have found a lot of patents. But you wouldn't have found this one until you told somebody, "Oh, I'm working on a spreadsheet," and he said, "Oh, did you know those other companies that are making spreadsheets are getting sued?" Then you would have found out.

Well, you can't find all the patents by searching, but you can find a lot of them. And then you've got to figure out what they mean. which is hard. because patents are

written in tortuous legal language which is very hard to understand the real meaning of. So you're going to have to spend a lot of time talking with an expensive lawyer explaining what you want to do in order to find out from the lawyer whether you're allowed to do it.

Even the patent holders often can't recognize just what their patents mean. For instance, there's somebody named Paul Heckel who released a program for displaying a lot of data on a small screen, and based on a couple of the ideas in that program he got a couple of patents.

I once tried to find a simple way to describe what claim 1 of one of those patents covered. I found that I couldn't find any simpler way of saying it than what was in the patent itself, and that sentence, I couldn't manage to keep it all in my mind at once, no matter how hard I tried.

And Heckel couldn't follow it either, because when he saw HyperCard, all he noticed was it was nothing like his program. It didn't occur to him that the way his patent

was written it might prohibit HyperCard; but his lawyer had that idea, so he threatened Apple. And then he threatened Apple's customers, and eventually Apple made a settlement with him which is secret, so we don't know who really won. And this is just an illustration of how hard it is for anybody to understand what a patent does or doesn't prohibit.

In fact, I once gave this speech and Heckel was in the audience. And at this point he jumped up and said, "That's not true, I just didn't know the scope of my protection." And I said, "Yeah, that's what I said," at which point he sat down and that was the end of my experience being heckled by Heckel. If I had said no, he probably would have found a way to argue with me.

Anyway, after a long, expensive conversation with a lawyer, the lawyer will give you an answer like this:

If you do something in this area, you're almost certain to lose a lawsuit; if you do something in this area, there's a considerable chance of losing a lawsuit; and if you really want to be safe you've got to stay out of this area. But there's a sizeable element of chance in the outcome of any lawsuit.

So now that you have clear, predictable rules for doing business, what are you actually going to do? Well, there are three things that you could do to deal with the issue of any particular patent. One is to avoid it, another is to get a license for it, and the third is to invalidate it. So I'll talk about these one by one.

First, there's the possibility of avoiding the patent, which means, don't implement what it prohibits. Of course, if it's hard to tell what it prohibits, it might be hard to tell what would suffice to avoid it.

A couple of years ago Kodak sued Sun [for] using a patent for something having to do with object-oriented programming, and Sun didn't think it was infringing that patent. But the court decided it was; and when other people look at that patent they haven't the faintest idea whether that decision was right or not. No one can tell what that patent does or doesn't cover, but Sun had to pay hundreds of millions of dollars because of violating a completely incomprehensible law.

Sometimes you can tell what you need to avoid, and

sometimes what you need to avoid is an algorithm.

For instance, I saw a patent for something like the fast Fourier transform, but it ran twice as fast. Well, if the ordinary FFT is fast enough for your application then that's an easy way to avoid this other one. And most of the time that would work. Once in a while you might be trying to do something where it runs doing FFT all the time, and it's just barely fast enough using the faster algorithm. And then you can't avoid it, although maybe you could wait a couple of years for a faster computer. But that's going to be rare. Most of the time that patent will to be easy to avoid.

On the other hand, a patent on an algorithm may be impossible to avoid. Consider the LZW data compression algorithm. Well, as I explained, we found a better data compression algorithm, and everybody who wanted to compress files switched to the program `gzip` which used the better algorithm. And the reason is, if you just want to compress the file and uncompress it later, you can tell people to use this program to uncompress it; then you can use any program with any algorithm, and you only care how well it works

you only care how well it works.

But LZW is used for other things, too; for instance the PostScript language specifies operators for LZW compression and LZW uncompression. It's no use having another, better algorithm because it makes a different format of data. They're not interoperable. If you compress it with the `gzip` algorithm, you won't be able to uncompress it using LZW. So no matter how good your other algorithm is, and no matter what it is, it just doesn't enable you to implement PostScript according to the specs.

But I noticed that users rarely ask their printers to compress things. Generally the only thing they want their printers to do is to uncompress; and I also noticed that both of the patents on the LZW algorithm were written in such a way that if your system can only uncompress, it's not forbidden. These patents were written so that they covered compression, and they had other claims covering both compression and uncompression; but there was no claim covering only uncompression. So I realized that if we implement only the uncompression for LZW, we would be safe. And although it would not satisfy the

specification, it would please the users sufficiently; it would do what they actually needed. So that's how we barely squeaked by avoiding the two patents.

Now there is gif format, for images. That uses the LZW algorithm also. It didn't take long for people to define another image format, called png, which stands for "Png's Not Gif." I think it uses the gzip algorithm. And we started saying to people, "Don't use gif format, it's dangerous. Switch to png." And the users said, "Well, maybe some day, but the browsers don't implement it yet," and the browser developers said, "We may implement it someday, but there's not much demand from users."

Well, it's pretty obvious what's going on—gif was a de facto standard. In effect, asking people to switch to a different format, instead of their de facto standard, is like asking everyone in New Zealand to speak Hungarian. People will say, "Well, yeah, I'll learn to speak it after everyone else does." And so we never succeeded in asking people to stop using gif, even though one of those patent holders was going around to operators of web sites, threatening to sue them unless they could agree that

sites, threatening to sue them unless they could prove that all of the gifs on the site were made with authorized, licensed software.

So gif was a dangerous trap for a large part of our community. We thought we had an alternative to gif format, namely jpeg, but then somebody said, “I was just looking through my portfolio of patents”—I think it was somebody that just bought patents and used them to threaten people—and he said, “and I found that one of them covers jpeg format.”

Well, jpeg was not a de facto standard, it’s an official standard, issued by a standards committee; and the committee had a lawyer too. Their lawyer said he didn’t think that this patent actually covered jpeg format.

So who’s right? Well, this patent holder sued a bunch of companies, and if there was a decision, it would have said who was right. But I haven’t heard about a decision; I’m not sure if there ever was one. I think they settled, and the settlement is almost certainly secret, which means that it didn’t tell us anything about who’s right.

These are fairly lightweight cases: one patent on jpeg, two patents on the LZW algorithm used in gif. Now you might wonder how come there are two patents on the same algorithm? It's not supposed to happen, but it did. And the reason is that the patent examiners can't possibly take the time to study every pair of things they might need to study and compare, because they're not allowed to take that much time. And because algorithms are just mathematics, there's no way you can narrow down which applications and patents you need to compare.

You see, in physical engineering fields, they can use the physical nature of what's going on to narrow things down. For instance, in chemical engineering, they can say, "What are the substances going in? What are the substances coming out?" If two different [patent] applications are different in that way, then they're not the same process so you don't need to worry. But the same math can be represented in ways that can look very different, and until you study them both together, you don't realize they're talking about the same thing. And, because of this, it's quite common to see the same thing

because of this, it's quite common to see the same thing get patented multiple times [in software].

Remember that program that was killed by a patent before we released it? Well, that algorithm got patented twice also. In one little field we've seen it happen in two cases that we ran into—the same algorithm being patented twice. Well, I think my explanation tells you why that happens.

But one or two patents is a lightweight case. What about mpeg2, the video format? I saw a list of over 70 patents covering that, and the negotiations to arrange a way for somebody to license all those patents took longer than developing the standard itself. The jpeg committee wanted to develop a follow-on standard, and they gave up. They said there were too many patents; there was no way to do it.

Sometimes it's a feature that's patented, and the only way to avoid that patent is not to implement that feature. For instance, the users of the word processor Xywrite once got a downgrade in the mail, which removed a feature. The feature was that you could define a list of

abbreviations. For instance, if you define “exp” as an abbreviation for “experiment,” then if you type “exp-space” or “exp-comma,” the “exp” would change automatically to “experiment.”

Then somebody who had a patent on this feature threatened them, and they concluded that the only thing they could do was to take the feature out. And so they sent all the users a downgrade.

But they also contacted me, because my Emacs editor had a feature like that starting from the late 70s. And it was described in the Emacs manual, so they thought I might be able to help them invalidate that patent. Well, I’m happy to know I’ve had at least one patentable idea in my life, but I’m unhappy that someone else patented it.

Fortunately, in fact, that patent was eventually invalidated, and partly on the strength of the fact that I had published using it earlier. But in the meantime they had had to remove this feature.

Now, to remove one or two features may not be a

disaster. But when you have to remove 50 features, you could do it, but people are likely to say, “This program’s no good; it’s missing all the features I want.” So it may not be a solution. And sometimes a patent is so broad that it wipes out an entire field, like the patent on public-key encryption, which in fact put public-key encryption basically off limits for about ten years.

So that’s the option of avoiding the patent—often possible, but sometimes not, and there’s a limit to how many patents you can avoid.

What about the next possibility, of getting a license for the patent?

Well, the patent holder may not offer you a license. It’s entirely up to him. He could say, “I just want to shut you down.” I once got a letter from somebody whose family business was making casino games, which were of course computerized, and he had been threatened by a patent holder who wanted to make his business shut down. He sent me the patent. Claim 1 was something like “a network with a multiplicity of computers, in which

each computer supports a multiplicity of games, and allows a multiplicity of game sessions at the same time.”

Now, I'm sure in the 1980s there was a university that set up a room with a network of workstations, and each workstation had some kind of windowing facility. All they had to do was to install multiple games and it would be possible to display multiple game sessions at once. This is so trivial and uninteresting that nobody would have bothered to publish an article about doing it. No one would have been interested in publishing an article about doing it, but it was worth patenting it. If it had occurred to you that you could get a monopoly on this trivial thing, then you could shut down your competitors with it.

But why does the Patent Office issue so many patents that seem absurd and trivial to us?

It's not because the patent examiners are stupid, it's because they're following a system, and the system has rules, and the rules lead to this result.

You see, if somebody has made a machine that does

something once, and somebody else designs a machine that will do the same thing, but N times, for us that's a `for-loop`, but for the Patent Office that's an invention. If there are machines that can do A, and there are machines that can do B, and somebody designs a machine that can do A or B, for us that's an `if-then-else` statement, but for the Patent Office that's an invention. So they have very low standards, and they follow those standards; and the result is patents that look absurd and trivial to us. Whether they're legally valid I can't say. But every programmer who sees them laughs.

In any case, I was unable to suggest anything he could do to help himself, and he had to shut down his business. But most patent holders will offer you a license. It's likely to be rather expensive.

But there are some software developers that find it particularly easy to get licenses, most of the time. Those are the megacorporations. In any field the megacorporations generally own about half the patents, and they cross-license each other, and they can make anybody else cross-license if he's really producing anything. The result is that they end up peacefully with

anything. The result is that they end up painlessly with licenses for almost all the patents.

IBM wrote an article in its house magazine, *Think* magazine—I think it's issue 5, 1990—about the benefit IBM got from its almost 9,000 US patents at the time (now it's up to 45,000 or more). They said that one of the benefits was that they collected money, but the main benefit, which they said was perhaps an order of magnitude greater, was “getting access to the patents of others,” namely cross-licensing.

What this means is since IBM, with so many patents, can make almost everybody give them a cross-license, IBM avoids almost all the grief that the patent system would have inflicted on anybody else. So that's why IBM wants software patents. That's why the megacorporations in general want software patents, because they know that by cross-licensing, they will have a sort of exclusive club on top of a mountain peak. And all the rest of us will be down here, and there's no way we can get up there. You know, if you're a genius, you might start up a small company and get some patents, but you'll never get into IBM's league, no matter what you

do.

Now a lot of companies tell their employees, “Get us patents so we can defend ourselves” and they mean, “use them to try to get cross-licensing,” but it just doesn’t work well. It’s not an effective strategy if you’ve got a small number of patents.

Suppose you’ve got three patents. One points there, one points there, and one points there, and somebody over there points a patent at you. Well, your three patents don’t help you at all, because none of them points at him. On the other hand, sooner or later, somebody in the company is going to notice that this patent is actually pointing at some people, and [the company] could threaten them and squeeze money out of them—never mind that those people didn’t attack this company.

So if your employer says to you, “We need some patents to defend ourselves, so help us get patents,” I recommend this response:

Boss, I trust you and I’m sure you would only use those

patents to defend the company if it's attacked. But I don't know who's going to be the CEO of this company in five years. For all I know, it might get acquired by Microsoft. So I really can't trust the company's word to only use these patents for defense unless I get it in writing. Please put it in writing that any patents I provide for the company will only be used for self-defense and collective security, and not for repression, and then I'll be able to get patents for the company with a clean conscience.

It would be most interesting to raise this not just in private with your boss, but also on the company's discussion list.

The other thing that could happen is that the company could fail and its assets could be auctioned off, including the patents; and the patents will be bought by someone who means to use them to do something nasty.

This cross-licensing practice is very important to understand, because this is what punctures the argument of the software patent advocates who say that software patents are needed to protect the starving genius. They give you a scenario which is a series of unlikelyhoods.

So let's look at it. According to this scenario, there's a brilliant designer of whatever, who's been working for years by himself in his attic coming up with a better way to do whatever it is. And now that it's ready, he wants to start a business and mass-produce this thing; and because his idea is so good his company will inevitably succeed—except for one thing: the big companies will compete with him and take all his market the away. And because of this, his business will almost certainly fail, and then he will starve.

Well, let's look at all the unlikely assumptions here.

First of all, that he comes up with this idea working by himself. That's not very likely. In a high-tech field, most progress is made by people working in a field, doing things and talking with people in the field. But I wouldn't say it's impossible, not that one thing by itself.

But anyway the next supposition is that he's going to start a business and that it's going to succeed. Well, just because he's a brilliant engineer doesn't mean that he's any good at running a business. Most new businesses fail; more than 95 percent of them, I think, fail within a few

more than 95 percent of them, I think, fail within a few years. So that's probably what's going to happen to him, no matter what.

Ok, let's assume that in addition to being a brilliant engineer who came up with something great by himself, he's also talented at running businesses. If he has a knack for running businesses, then maybe his business won't fail. After all, not all new businesses fail, there are a certain few that succeed. Well, if he understands business, then instead of trying to go head to head with large companies, he might try to do things that small companies are better at and have a better chance of succeeding. He might succeed. But let's suppose it fails anyway. If he's so brilliant and has a knack for running businesses, I'm sure he won't starve, because somebody will want to give him a job.

So a series of unlikelyhoods—it's not a very plausible scenario. But let's look at it anyway.

Because where they go from there is to say the patent system will "protect" our starving genius, because he can get a patent on this technique. And then when

IBM wants to compete with him, he says, “IBM, you can’t compete with me, because I’ve got this patent,” and IBM says, “Oh, no, not again!”

Well, here’s what really happens.

IBM says, “Oh, how nice, you have a patent. Well, we have this patent, and this patent, and this patent, and this patent, and this patent, all of which cover other ideas implemented in your product, and if you think you can fight us on all those, we’ll pull out some more. So let’s sign a cross-license agreement, and that way nobody will get hurt.” Now since we’ve assumed that our genius understands business, he’s going to realize that he has no choice. He’s going to sign the cross-license agreement, as just about everybody does when IBM demands it. And then this means that IBM will get “access” to his patent, meaning IBM would be free to compete with him just as if there were no patents, which means that the supposed benefit that they claim he would get by having this patent is not real. He won’t get this benefit.

The patent might “protect” him from competition from you or me—but not from IBM—not from the very

from you or IBM, can not from IBM — not from the very megacorporations which the scenario says are the threat to him. You know in advance that there's got to be a flaw in this reasoning when people who are lobbyists for megacorporations recommend a policy supposedly because it's going to protect their small competitors from them. If it really were going to do that, they wouldn't be in favor of it. But this explains why [software patents] won't do it.

Even IBM can't always do this, because there are companies that we refer to as patent trolls or patent parasites, and their only business is using patents to squeeze money out of people who really make something.

Patent lawyers tell us that it's really wonderful to have patents in your field, but they don't have patents in their field. There are no patents on how to send or write a threatening letter, no patents on how to file a lawsuit, and no patents on how to persuade a judge or jury, so even IBM can't make the patent trolls cross-license. But IBM figures, "Our competition will have to pay them too; this is just part of the cost of doing business, and we can

live with it.” IBM and the other megacorporations figure that the general dominion over all activity that they get from their patents is good for them, and paying off the trolls they can live with. So that’s why they want software patents.

There are also certain software developers who find it particularly difficult to get a patent license, and those are the developers of free software. The reason is that the usual patent license has conditions we can’t possibly fulfill, because usual patent licenses demand a payment per copy. But when software gives users the freedom to distribute and make more copies, we have no way to count the copies that exist.

If someone offered me a patent license for a payment of one-millionth of a dollar per copy, the total amount of money I’d have to pay maybe is in my pocket now. Maybe it’s \$50, but I don’t know if it’s \$50, or \$49, or what, because there’s no way I can count the copies that people have made.

A patent holder doesn’t have to demand a payment per copy: a patent holder could offer you a license for a

per copy, a patent holder could offer you a license for a single lump sum, but those lump sums tend to be big, like US\$100,000.

And the reason that we've been able to develop so much freedom-respecting software is [that] we can develop software without money, but we can't pay a lot of money without money. If we're forced to pay for the privilege of writing software for the public, we won't be able to do it very much.

That's the possibility of getting a license for the patent. The other possibility is to invalidate the patent. If the country considers software patents to be basically valid, and allowed, the only question is whether that particular patent meets the criteria. It's only useful to go to court if you've got an argument to make that might prevail.

What would that argument be? You have to find evidence that, years ago, before the patent was applied for, people knew about the same idea. And you'd have to find things today that demonstrate that they knew about it publicly at that time. So the dice were cast years

ago, and if they came up favorably for you, and if you can prove that fact today, then you have an argument to use to try to invalidate the patent. And it might work.

It might cost you a lot of money to go through this case, and as a result, a probably invalid patent is a very frightening weapon to be threatened with if you don't have a lot of money. There are people who can't afford to defend their rights—lots of them. The ones who can afford it are the exception.

These are the three things that you might be able to do about each patent that prohibits something in your program. The thing is, whether each one is possible depends on different details of the circumstances, so some of the time, none of them is possible; and when that happens, your project is dead.

But lawyers in most countries tell us, “Don't try to find the patents in advance,” and the reason is that the penalty for infringement is bigger if you knew about the patent. So what they tell you is “Keep your eyes shut. Don't try to find out about the patents, just go blindly taking your design decisions. and hope.”

And of course, with each single design decision, you probably don't step on a patent. Probably nothing happens to you. But there are so many steps you have to take to get across the minefield, it's very unlikely you will get through safely. And of course, the patent holders don't all show up at the same time, so you don't know how many there are going to be.

The patent holder of the natural order recalculation patent was demanding 5 percent of the gross sales of every spreadsheet. You could imagine paying for a few such licenses, but what happens when patent holder number 20 comes along, and wants you to pay out the last remaining 5 percent? And then what happens when patent holder number 21 comes along?

People in business say that this scenario is amusing but absurd, because your business would fail long before you got there. They told me that two or three such licenses would make your business fail. So you'd never get to 20. They show up one by one, so you never know how many more there are going to be.

Software patents are a mess. They're a mess for software developers, but in addition they're a restriction on every computer user because software patents restrict what you can do on your computer.

This is very different from patents, for instance, on automobile engines. These only restrict companies that make cars; they don't restrict you and me. But software patents do restrict you and me, and everybody who uses computers. So we can't think of them in purely economic terms; we can't judge this issue purely in economic terms. There's something more important at stake.

But even in economic terms, the system is self-defeating, because its purpose is supposed to be to promote progress. Supposedly by creating this artificial incentive for people to publish ideas, it's going to help the field progress. But all it does is the exact opposite, because the big job in software is not coming up with ideas, it's implementing thousands of ideas together in one program. And software patents obstruct that, so they're economically self-defeating.

And there's even economic research showing that this is so—showing how in a field with a lot of incremental innovation, a patent system can actually reduce investment in R & D. And of course, it also obstructs development in other ways. So even if we ignore the injustice of software patents, even if we were to look at it in the narrow economic terms that are usually proposed, it's still harmful.

People sometimes respond by saying that “People in other fields have been living with patents for decades, and they've gotten used to it, so why should you be an exception?”

Now, that question has an absurd assumption. It's like saying, “Other people get cancer, why shouldn't you?” I think every time someone doesn't get cancer, that's good, regardless of what happened to the others. That question is absurd because of its presupposition that somehow we all have a duty to suffer the harm done by patents.

But there is a sensible question buried inside it, and that sensible question is “What differences are there

that better question is “What differences are there between various fields that might affect what is good or bad patent policy in those fields?”

There is an important basic difference between fields in regard to how many patents are likely to prohibit or cover parts of any one product.

Now we have a naive idea in our minds which I’m trying to get rid of, because it’s not true. And it’s that on any one product there is one patent, and that patent covers the overall design of that product. So if you design a new product, it can’t be patented already, and you will have an opportunity to get “the patent” on that product.

That’s not how things work. In the 1800s, maybe they did, but not now. In fact, fields fall on a spectrum of how many patents [there are] per product. The beginning of the spectrum is one, but no field is like that today; fields are at various places on this spectrum.

The field that’s closest to that is pharmaceuticals. A few decades ago, there really was one patent per pharmaceutical, at least at any time, because the patent

pharmaceutical, at least at any time, because the patent covered the entire chemical formula of that one particular substance. Back then, if you developed a new drug, you could be sure it wasn't already patented by somebody else and you could get the one patent on that drug.

But that's not how it works now. Now there are broader patents, so now you could develop a new drug, and you're not allowed to make it because somebody has a broader patent which covers it already.

And there might even be a few such patents covering your new drug simultaneously, but there won't be hundreds. The reason is, our ability to do biochemical engineering is so limited that nobody knows how to combine so many ideas to make something that's useful in medicine. If you can combine a couple of them you're doing pretty well at our level of knowledge. But other fields involve combining more ideas to make one thing.

At the other end of the spectrum is software, where we can combine more ideas into one usable design than anybody else, because our field is basically easier than all other fields. I'm presuming that the intelligence of people

in our field is the same as that of people in physical engineering. It's not that we're fundamentally better than they are; it's that our field is fundamentally easier, because we're working with mathematics.

A program is made out of mathematical components, which have a definition, whereas physical objects don't have a definition. The matter does what it does, so through the perversity of matter, your design may not work the way it "should" have worked. And that's just tough. You can't say that the matter has a bug in it, and the physical universe should get fixed. [Whereas] we [programmers] can make a castle that rests on a mathematically thin line, and it stays up because nothing weighs anything.

There're so many complications you have to cope with in physical engineering that we don't have to worry about.

For instance, when I put an `if`-statement inside of a `while`-loop,

- I don't have to worry that if this `while`-loop

repeats at the wrong rate, the `if`-statement might start to vibrate and it might resonate and crack;

- I don't have to worry that if it resonates much faster—you know, millions of times per second—that it might generate radio frequency signals that might induce wrong values in other parts of the program;
- I don't have to worry that corrosive fluids from the environment might seep in between the `if`-statement and the `while`-statement and start eating away at them until the signals don't pass anymore;
- I don't have to worry about how the heat generated by my `if`-statement is going to get out through the `while`-statement so that it doesn't make the `if`-statement burn out; and
- I don't have to worry about how I would take out the broken `if`-statement if it does crack, burn, or corrode, and replace it with another `if`-statement to make the program run again.

For that matter, I don't have to worry about how I'm going to insert the `if`-statement inside the `while`-

statement every time I produce a copy of the program. I don't have to design a factory to make copies of my program, because there are various general commands that will make copies of anything.

If I want to make copies on CD, I just have to write a master; and there's one program I can [use to] make a master out of anything, write any data I want. I can make a master CD and write it and send it off to a factory, and they'll duplicate whatever I send them. I don't have to design a different factory for each thing I want to duplicate.

Very often with physical engineering you have to do that; you have to design products for manufacturability. Designing the factory may even be a bigger job than designing the product, and then you may have to spend millions of dollars to build the factory. So with all of this trouble, you're not going to be able to put together so many different ideas in one product and have it work.

A physical design with a million nonrepeating different design elements is a gigantic project. A program with a million different design elements that's nothing

with a million different design elements, that's troubling. It's a few hundred thousand lines of code, and a few people will write that in a few years, so it's not a big deal. So the result is that the patent system weighs proportionately heavier on us than it does on people in any other field who are being held back by the perversity of matter.

A lawyer did a study of one particular large program, namely the kernel Linux, which is used together with the GNU operating system that I launched. This was five years ago now; he found 283 different US patents, each of which appeared to prohibit some computation done somewhere in the code of Linux. At the time I saw an article saying that Linux was 0.25 percent of the whole system. So by multiplying 300 by 400 we can estimate the number of patents that would prohibit something in the whole system as being around 100,000. This is a very rough estimate only, and no more accurate information is available, since trying to figure it out would be a gigantic task.

Now this lawyer did not publish the list of patents, because that would have endangered the developers of

Linux the kernel, putting them in a position where the penalties if they were sued would be greater. He didn't want to hurt them; he wanted to demonstrate how bad this problem is, of patent gridlock.

Programmers can understand this immediately, but politicians usually don't know much about programming; they usually imagine that patents are basically much like copyrights, only somehow stronger. They imagine that since software developers are not endangered by the copyrights on their work, that they won't be endangered by the patents on their work either. They imagine that, since when you write a program you have the copyright, [therefore likewise] if you write a program you have the patents also. This is false—so how do we give them a clue what patents would really do? What they really do in countries like the US?

I find it's useful to make an analogy between software and symphonies. Here's why it's a good analogy.

A program or symphony combines many ideas. A symphony combines many musical ideas. But you can't

Symphony combines many musical ideas. But you can't just pick a bunch of ideas and say "Here's my combination of ideas, do you like it?" Because in order to make them work you have to implement them all. You can't just pick musical ideas and list them and say, "Hey, how do you like this combination?" You can't hear that [list]. You have to write notes which implement all these ideas together.

The hard task, the thing most of us wouldn't be any good at, is writing all these notes to make the whole thing sound good. Sure, lots of us could pick musical ideas out of a list, but we wouldn't know how to write a good-sounding symphony to implement those ideas. Only some of us have that talent. That's the thing that limits you. I could probably invent a few musical ideas, but I wouldn't know how to use them to any effect.

So imagine that it's the 1700s, and the governments of Europe decide that they want to promote the progress of symphonic music by establishing a system of musical idea patents, so that any musical idea described in words could be patented.

For instance, using a particular sequence of notes as a motif could be patented, or a chord progression could be patented, or a rhythmic pattern could be patented, or using certain instruments by themselves could be patented, or a format of repetitions in a movement could be patented. Any sort of musical idea that could be described in words would have been patentable.

Now imagine that it's 1800 and you're Beethoven, and you want to write a symphony. You're going to find it's much harder to write a symphony you don't get sued for than to write one that sounds good, because you have to thread your way around all the patents that exist. If you complained about this, the patent holders would say, "Oh, Beethoven, you're just jealous because we had these ideas first. Why don't you go and think of some ideas of your own?"

Now Beethoven had ideas of his own. The reason he's considered a great composer is because of all of the new ideas that he had, and he actually used. And he knew how to use them in such a way that they would work, which was to combine them with lots of well-known ideas. He could put a few new ideas into a

known ideas. He could put a few new ideas into a composition together with a lot of old and uncontroversial ideas. And the result was a piece that was controversial, but not so much so that people couldn't get used to it.

To us, Beethoven's music doesn't sound controversial; I'm told it was, when it was new. But because he combined his new ideas with a lot of known ideas, he was able to give people a chance to stretch a certain amount. And they could, which is why to us those ideas sound just fine. But nobody, not even a Beethoven, is such a genius that he could reinvent music from zero, not using any of the well-known ideas, and make something that people would want to listen to. And nobody is such a genius he could reinvent computing from zero, not using any of the well-known ideas, and make something that people want to use.

When the technological context changes so frequently, you end up with a situation where what was done 20 years ago is totally inadequate. Twenty years ago there was no World Wide Web. So, sure, people did a lot of things with computers back then, but what

they want to do today are things that work with the World Wide Web. And you can't do that using only the ideas that were known 20 years ago. And I presume that the technological context will continue to change, creating fresh opportunities for somebody to get patents that give the shaft to the whole field.

Big companies can even do this themselves. For instance, a few years ago Microsoft decided to make a phony open standard for documents and to get it approved as a standard by corrupting the International Standards Organization, which they did. But they designed it using something that Microsoft had patented. Microsoft is big enough that it can start with a patent, design a format or protocol to use that patented idea (whether it's helpful or not), in such a way that there's no way to be compatible unless you use that same idea too. And then Microsoft can make that a de facto standard with or without help from corrupted standards bodies. Just by its weight it can push people into using that format, and that basically means that they get a stranglehold over the whole world. So we need to show the politicians what's really going on here. We need to

show them why this is bad.

Now I've heard it said that the reason New Zealand is considering software patents is that one large company wants to be given some monopolies. To restrict everyone in the country so that one company will make more money is the absolute opposite of statesmanship.

Microsoft's New Monopoly

This essay was originally published on <http://gnu.org>, in 2005. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

This article was written in July 2005. Microsoft adopted a different policy in 2006, so the specific policies described below and the specific criticisms of them are only of historical significance. The overall problem remains, however: Microsoft's cunningly worded new policy (see

<http://grokdoc.net/index.php/OOXML/objections%28Patent%20rights%20to%20implement%20the%20Ecml%20376%29%20specification%20have%20not%20been%20granted%29>
does not give anyone clear permission to implement OOXML.

European legislators who endorse software patents frequently claim that those wouldn't affect free software (or "open source"). Microsoft's lawyers are determined to prove they are mistaken.

Leaked internal documents in 1998 said that Microsoft considered the free software GNU/Linux operating system (referred to therein as "Linux") as the principal competitor to Windows, and spoke of using patents and secret file formats to hold us back.

Because Microsoft has so much market power, it can often impose new standards at will. It need only patent some minor idea, design a file format, programming language, or communication protocol based on it, and then pressure users to adopt it. Then we in the free software community will be forbidden to provide software that does what these users want; they will be locked in to Microsoft, and we will be locked out from serving them.

Previously Microsoft tried to get its patented scheme for spam blocking adopted as an Internet standard, so as to exclude free software from handling email. The standards committee in charge rejected the proposal, but Microsoft said it would try to convince large ISPs to use the scheme anyway.

Now Microsoft is planning to try something similar for Word files.

Several years ago, Microsoft abandoned its documented format for saving documents, and switched to a new format which was secret. However, the developers of free software word processors such as AbiWord and OpenOffice.org experimented assiduously for years to figure out the format, and now those programs can read most Word files. But Microsoft isn't licked yet.

The next version of Microsoft Word will use formats that involve a technique that Microsoft claims to hold a patent on. Microsoft offers a royalty-free patent license for certain limited purposes, but it is so limited that it does not allow free software. You can see the license here: <http://microsoft.com/whdc/xps/xpspatentlic.mspx>.

Free software is defined as software that respects four fundamental freedoms: (0) freedom to run the software as you wish, (1) freedom to study the source code and modify it to do what you wish, (2) freedom to make and redistribute copies, and (3) freedom to publish modified versions. Only programmers can directly exercise freedoms 1 and 3, but all users can exercise freedoms 0 and 2, and all users benefit from the modifications that programmers write and publish.

Distributing an application under Microsoft's patent license imposes license terms that prohibit most possible modifications of the software. Lacking freedom 3, the freedom to publish modified versions, it would not be free software. (I think it could not be "open source" software either, since that definition is similar; but it is not identical, and I cannot speak for the advocates of open source.)

The Microsoft license also requires inclusion of a specific statement. That requirement would in itself prevent the program from being free: it is normal for free software to carry license notices that cannot be changed, and this statement could be included in one of them. The statement is biased and confusing, since it uses the term "intellectual property"; fortunately, one is not required to endorse the statement as true or even meaningful, only to include it. The software developer could cancel its misleading effect with a disclaimer like this: "The following misleading statement has been imposed on us by Microsoft; please be advised that it is propaganda. See <http://gnu.org/philosophy/not-ipr.html> for more explanation."

However, the requirement to include a fixed piece of text is actually quite cunning, because anyone who does so has explicitly accepted and applied the restrictions of the Microsoft patent license. The resulting program is clearly not free software.

Some free software licenses, such as the most popular GNU General Public License (GNU GPL), forbid publication of a modified version if it isn't free software in the same way. (We call that the "liberty or death" clause, since it ensures the program will remain free or die.) To apply Microsoft's license to a program under the GNU GPL would violate the program's license; it would be illegal. Many other free software licenses permit nonfree modified versions. It wouldn't be illegal to modify such a program and publish the modified version under Microsoft's patent license. But that modified version, with its modified license, wouldn't be free software.

Microsoft's patent covering the new Word format is a US patent. It doesn't restrict anyone in Europe; Europeans are free to make and use software that can read this format. Europeans that develop or use software currently enjoy an advantage over Americans: Americans can be sued for patent infringement for their software activities in the US, but the Europeans cannot be sued for their activities in Europe. Europeans can already get US software patents and sue Americans, but Americans cannot get European software patents if Europe doesn't allow them.

All that will change if the European Parliament authorizes software patents. Microsoft will be one of thousands of foreign software patent holders that will bring their patents over to Europe to sue the software developers and computer users there. Of the 50,000-odd putatively invalid software patents issued by the European Patent Office, around 80 percent do not belong to Europeans. The European Parliament should vote to keep these patents invalid, and keep Europeans safe.

2009 Note

The EU directive to allow software patents was rejected, but the European Patent Office has continued issuing them and some countries treat them as valid. See <http://effi.org> for more information and to participate in the campaign against software patents in Europe.

Part V

The Licenses

Chapter 27

Introduction to the Licenses

Copyright © 2010 Free Software Foundation, Inc.

This essay is published in *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Written by Brett Smith and Richard Stallman.

This part contains the text of the latest versions of the primary GNU licenses: the GNU General Public License (GNU GPL), the GNU Lesser General Public License (LGPL), and the GNU Free Documentation License (FDL). Though they are legal documents, they belong in this book of essays because they are concrete expressions of the ideals of free software.

Software development for the GNU operating

system began in 1984. Once Richard Stallman had parts of the GNU system that were worth releasing, he needed a license to release them under. Some free software licenses already existed; these gave users permission to modify and redistribute the software, but they also allowed using the code in proprietary versions and proprietary programs. Using those licenses, GNU would have failed to achieve its goal of delivering freedom to all users, because middlemen would have converted the GNU code into proprietary software.

So Stallman devised a license to assure every user the freedom to modify and redistribute the software. It granted these permissions under one key condition: whoever distributed the software must pass along the authorization to modify and redistribute that same software, along with the source code making it practical to do so. Stallman coined the term “copyleft” (see “What Is Copyleft?” on [\(195\)](#)) to describe this key twist of using the legal power of copyright to ensure freedom for all users.

GNU copyleft licenses were first developed for

software, and later for related areas such as software documentation. In them, the principles of the free software movement, explained throughout the essays in this book, take practical form. Each of their successive revisions has had to wrestle with free software's legal and practical obstacles and offers numerous illustrations of how free software ideals are codified into legal terms.

The Origins of the GPL

The first version of the GNU General Public License was published in 1989—but Stallman had been releasing software under copyleft licenses as part of the GNU Project since as early as 1985. Prior to 1989, each published GNU program had been covered by a license specifically tailored for it. Instead of a single GNU General Public License, there was a GNU CC General Public License, a GDB General Public License, and so on. These licenses were identical except for minor differences: for instance, terms about displaying license notices to users were different for different programs and, unless it covered a program that was just one source file, each license contained the name of the program it applied to.

program it applied to.

By 1989, Stallman had had enough experience with different GNU packages under slightly different licenses to conclude that it was crucial to unify them into one license that would cover all these packages. He worked with Jerry Cohen, an attorney at Perkins Smith & Cohen LLP, to collect concepts from all the different licenses written up to that point, and bring them together into one license. It was thus that on 1 February 1989 the GNU General Public License was born.

The first version of the license sought to ensure two results: first, that all derived works of the software would be released under the same license and, second, that everyone who received the software would have a chance to get the source code. These requirements implement a strong copyleft by blocking the three main ways of making programs proprietary: with copyright, with end user license agreements, and by not distributing source code.

In comparison to the program-specific licenses that had preceded it, GPL version 1 featured few substantial

changes—the GPL was evolutionary, not revolutionary—but it made a big practical difference. Previously, developers who had wanted to copyleft a program had needed to tailor one of the existing licenses to that program. Many had not bothered. With the release of the GPL, those developers had a license they could use out of the box to provide all of their users with freedom to share and change the software. It was a powerful tool.

Version 2

After the 1981 US Supreme Court decision in *Diamond v. Diehr*, the US Patent and Trademark Office began issuing patents for software. Software patents threaten free software and proprietary software alike (see part IV in this book), and Stallman realized that they could subvert the copyleft in the GNU GPL.

By selectively issuing patent licenses, patent holders can arbitrarily control how the software under them is distributed or modified. A patent holder can give one party permission to resell the program, another permission to develop and use a modified version at her

company, and a third permission to do all the activities that the GPL itself allows. They can demand whatever they wish in exchange for these permissions. They have this power over any software that implements the patented idea, whether or not they have modified or distributed it themselves. This power threatens free software because third parties with patents can impose restrictions on free software users and developers.

If patent holders don't distribute or modify software, then a software license based on copyright like the GPL can't control their activities: they haven't done anything that requires permission under the license. But the software license can stop each of the program's distributors from entering limiting agreements with the patent holder. Enter GPL version 2: a new section in the license (sec. 7) explicitly says that if parties are subject to other legal agreements—such as a patent license—that contradict the GPL's terms, then the licensee must refrain from distributing the software at all. As a result, any party that wants to distribute or modify the software, and also obtain a patent license, must ensure that the terms of that license are consistent with all of the GPL's conditions: recipients of the software must receive it under the same

recipients of the software must receive it under the same terms, with no additional restrictions, and have the means to get the source code.

This new section protected the integrity of the distribution system for GPL-covered software. A fundamental principle of the license is that every licensee, from the most humble individual to the largest corporation, has the exact same rights to share and change the software. Patent holders who do not distribute the software themselves and selectively issues patent licenses could potentially interfere with this goal, splitting licensees into different groups however they see fit. Section 7 of GPL version 2 prevents this abuse.

The LGPL

The GPL worked well for the programming tools, utilities, and games that were released by the GNU Project in the early years; however, Stallman recognized that releasing the recently developed GNU C Library the same way could backfire. Aside from some extensions, the GNU C Library was to be a compatible replacement for the UNIX C Library, so any C program would be

able link with either one. If proprietary C programs were not allowed to use the GNU C Library, they would simply use the UNIX library. Being strict in this case would gain nothing.

Stallman decided to compromise with a modified copyleft: one that would protect the freedom of the library itself, but not that of the programs that use it. This idea was implemented in a license originally called the GNU Library General Public License, first published as version 2.0, in June 1991. The original LGPL stated Conditions like the GPL's—with an important exception: if someone else's program used the library only by referring to it as a library, that program's source could be distributed under license terms of the author's choosing. However, the executable made by combining the program and the library had to come with a copy of the LGPL and source code for the library, and provide some mechanism for users who have modified the library to update the executable to use their modified library.

How does a developer use the work as a library in order to take advantage of the special set of conditions

provided by LGPLv2? Think of a computer program as a series of instructions for doing a particular job: compiling or linking the program with a library provides the programmer with a means to say, “When the program gets to this point, get further instructions from the library, and come back here when those are done.” Libraries are commonly used in software development because they make the effort less repetitive and less error prone: programmers don’t have to reinvent the wheel—and perhaps introduce bugs in the process—every time they want to accomplish a particular task. Because libraries are so widely created and used, developers have the means to readily take advantage of the LGPL’s additional permissions.

Version 2.0 of the license worked as intended: in some situations, proprietary software developers chose to use an LGPL-covered library over a proprietary alternative, and users received the freedom to share and change that library. This did not produce an “ideal” outcome—where the user had complete control over the entire program—but in these cases the GPL would not have achieved that ideal outcome either. The LGPL assured the users some freedom where they would have

assured the users some freedom where they would have otherwise had none.

The name “Library GPL” led some free software developers to assume all libraries ought on principle to be licensed this way, but that was not the intent—when a free library has no proprietary competitor, releasing it under the GNU GPL can benefit free software. To avoid this unintended message, Stallman renamed this license to the Lesser General Public License, and incremented the version number to 2.1 to reflect the relatively minor changes in the text: the license sported a new preamble, a few wording clarifications, and allowed programs to make their calls to the library through special system facilities for shared libraries where those are available. The Lesser General Public License version 2.1 was released in February 1999.

The FDL

At the turn of the century, free software was growing much faster than it had been previously; the documentation, however, was not keeping pace. Stallman was concerned about this failure and wrote

about it in “Free Software Needs Free Documentation” ([91](#)).

While there are some similarities between software and documentation—they are both works that are meant for practical use—there are important differences in the ways they can be used. The GPL and the LGPL were not suitable for manuals.

For some time, GNU packages had been using an untitled, simple, ad hoc copyleft license for each manual. Since each manual’s license was different, text could not be copied from one manual to another. So Stallman wrote the GNU Free Documentation License, a copyleft license designed primarily for software documentation and other practical written works.

The FDL was first published in March 2000. The principles of the copyleft remain the same: everyone who receives a copy of the work should be able to modify and redistribute it. Where the FDL differs from the software licenses is in the details of its implementation: conditions about how to attribute the work and provide

“source code”—an editable version of the document—are different.

Version 3

During the 1990s, as free software became more popular, the GPL emerged as the clear copyleft license of choice for the community, and was adopted by the majority of free software projects; at the same time, however, proprietary developers had come up with methods of effectively denying users the freedoms that the GPL was meant to protect, without actually violating the GPL. In addition, there were other practices that the GPL did not handle conveniently. To deal with these issues called for an updated version of the license.

Around 2002, Stallman and others at the Free Software Foundation began discussing how to update the GPL, and the LGPL along with it. The FSF established a public review process, run with help from attorneys at the Software Freedom Law Center, to catch possible problems before actually releasing the new licenses. Committees of advisors from the community studied issues raised by public comments and reported the

issues raised by public comments and reported the various positions and arguments to Stallman, who decided what policy to adopt; then he wrote license text with advice and suggestions from the attorneys. The importance of the changes made are explained in “Why Upgrade to GPLv3” ([283](#)).

Version 3 used new terminology to promote uniform interpretations in different jurisdictions, and modified some requirements to fit new practices in the free software community. Beyond that, it introduced several new conditions to strengthen the copyleft and thereby the free software community as a whole. For instance, it

- blocked distributors from restricting users by building hardware that rejects the users’ modified versions (“tivoization”);
- allowed code to carry limited additional requirements, for compatibility with some other popular free software licenses;
- and strengthened patent requirements by providing clear terms to handle patent cross-licenses, which are common arrangements between large patent-holding companies.

Both GPLv3 and LGPLv3 included terms to address all of these issues, and were finally released on 29 June 2007. These licenses are the state of the art in copyleft, going farther than any other software license to protect users' freedom and bring about a world in harmony with the ideals expressed in this book.

Chapter 28

The GNU General Public License

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc.

<http://fsf.org/>

51 Franklin St., Floor 5, Boston, MA 02110-1335,
USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to

A public license is included to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a

program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for

individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

- **0. Definitions.**

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with

it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy.

Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user

EXERCISE. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

- **1. Source Code.**

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an

implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow

between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

- **2. Basic Permissions.**

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered

works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

- **3. Protecting Users' Legal Rights From Anti-Circumvention Law.**

No covered work shall be deemed part of an

effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

- **4. Conveying Verbatim Copies.**

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any copyright notices for the

this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

- **5. Conveying Modified Source Versions.**

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement

This requirement modifies the requirement in section 4 to “keep intact all notices”.

- c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a

larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

- **6. Conveying Non-Source Forms.**

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b. Convey the object code in, or embodied

in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

- c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

- d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and

Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial

whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source

conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special

password or key for unpacking, reading or copying.

- **7. Additional Terms.**

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional

permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d. Limiting the use for publicity purposes of

or limiting the use for publicity purposes or names of licensors or authors of the material; or

- e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document,

provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

- **8. Termination.**

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under

section 10.

- **9. Acceptance Not Required for Having Copies.**

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

- **10. Automatic Licensing of Downstream Recipients.**

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate

that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or

counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

- **11. Patents.**

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the

requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License,

through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients.

“Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all

license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

- **12. No Surrender of Others' Freedom.**

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

- **13. Use with the GNU Affero General Public**

License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

- **14. Revised Versions of this License.**

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain

number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

- **15. Disclaimer of Warranty.**

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

- **16. Limitation of Liability.**

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT

WRITING WILL AN FCOI RIGHT
HOLDER, OR ANY OTHER PARTY WHO
MODIFIES AND/OR CONVEYS THE
PROGRAM AS PERMITTED ABOVE, BE
LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL,
INCIDENTAL OR CONSEQUENTIAL
DAMAGES ARISING OUT OF THE USE OR
INABILITY TO USE THE PROGRAM
(INCLUDING BUT NOT LIMITED TO LOSS
OF DATA OR DATA BEING RENDERED
INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE
OF THE PROGRAM TO OPERATE WITH
ANY OTHER PROGRAMS), EVEN IF SUCH
HOLDER OR OTHER PARTY HAS BEEN
ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES.

- **17. Interpretation of Sections 15 and 16.**

If the disclaimer of warranty and limitation of
liability provided above cannot be given local legal
effect according to their terms, reviewing courts

shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program's name and a brief idea of

what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name or author
This program comes with ABSOLUTELY NO WARRANTY; for
details type 'show w'. This is free software, and you are
welcome to redistribute it under certain conditions; type
'show c' for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read

[http://www.gnu.org/philosophy/why-not-
lgpl.html](http://www.gnu.org/philosophy/why-not-lgpl.html).

Chapter 29

Why Upgrade to GPLv3

Copyright © 2007, 2009 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2007.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Version 3 of the GNU General Public License (GNU GPL) has been released, enabling free software packages to upgrade from GPL version 2. This article explains why upgrading the license is important.

First of all, it is important to note that upgrading is a choice. GPL version 2 will remain a valid license, and no disaster will happen if some programs remain under GPLv2 while others advance to GPLv3. These two licenses are incompatible, but that isn't a fundamental

problem.

When we say that GPLv2 and GPLv3 are incompatible, it means there is no legal way to combine code under GPLv2 with code under GPLv3 in a single program. This is because both GPLv2 and GPLv3 are copyleft licenses: each of them says, “If you include code under this license in a larger program, the larger program must be under this license too.” There is no way to make them compatible. We could add a GPLv2-compatibility clause to GPLv3, but it wouldn’t do the job, because GPLv2 would need a similar clause.

Fortunately, license incompatibility matters only when you want to link, merge or combine code from two different programs into a single program. There is no problem in having GPLv3-covered and GPLv2-covered programs side by side in an operating system. For instance, the TeX license and the Apache license are incompatible with GPLv2, but that doesn’t stop us from running TeX and Apache in the same system with Linux, Bash and GCC. This is because they are all separate programs. Likewise, if Bash and GCC move to GPLv3,

while Linux remains under GPLv2, there is no conflict.

Keeping a program under GPLv2 won't create problems. The reason to migrate is because of the existing problems that GPLv3 will address.

One major danger that GPLv3 will block is tivoization. Tivoization means certain “appliances” (which have computers inside) contain GPL-covered software that you can't effectively change, because the appliance shuts down if it detects modified software. The usual motive for tivoization is that the software has features the manufacturer knows people will want to change, and aims to stop people from changing them. The manufacturers of these computers take advantage of the freedom that free software provides, but they don't let you do likewise.

Some argue that competition between appliances in a free market should suffice to keep nasty features to a low level. Perhaps competition alone would avoid arbitrary, pointless misfeatures like “Must shut down between 1pm and 5pm every Tuesday,” but even so, a choice of masters isn't freedom. Freedom means *you* control what

your software does, not merely that you can beg or threaten someone else who decides for you.

In the crucial area of Digital Restrictions Management (DRM)—nasty features designed to restrict your use of the data in your computer—competition is no help, because relevant competition is forbidden. Under the Digital Millennium Copyright Act and similar laws, it is illegal, in the US and many other countries, to distribute DVD players unless they restrict the user according to the official rules of the DVD conspiracy (its web site is <http://www.dvdrca.org/>, but the rules do not seem to be published there). The public can't reject DRM by buying non-DRM players because none are available. No matter how many products you can choose from, they all have equivalent digital handcuffs.

GPLv3 ensures you are free to remove the handcuffs. It doesn't forbid DRM, or any kind of feature. It places no limits on the substantive functionality you can add to a program, or remove from it. Rather, it makes sure that you are just as free to remove nasty features as the distributor of your copy was to add them. Tivoization

is the way they deny you that freedom; to protect your freedom, GPLv3 forbids tivoization.

The ban on tivoization applies to any product whose use by consumers is to be expected, even occasionally. GPLv3 tolerates tivoization only for products that are almost exclusively meant for businesses and organizations.

Another threat that GPLv3 resists is that of patent deals like the Novell-Microsoft pact. Microsoft wants to use its thousands of patents to make users pay Microsoft for the privilege of running GNU/Linux, and made this pact to try to achieve that. The deal offers rather limited protection from Microsoft patents to Novell's customers.

Microsoft made a few mistakes in the Novell-Microsoft deal, and GPLv3 is designed to turn them against Microsoft, extending that limited patent protection to the whole community. In order to take advantage of this protection, programs need to use GPLv3.

Microsoft's lawyers are not stupid, and next time they may manage to avoid those mistakes. GPLv3

therefore says they don't get a "next time." Releasing a program under GPL version 3 protects it from Microsoft's future attempts to make redistributors collect Microsoft royalties from the program's users.

GPLv3 also provides users with explicit patent protection from the program's contributors and redistributors. With GPLv2, users rely on an implicit patent license to make sure that the company which provided them a copy won't sue them, or the people they redistribute copies to, for patent infringement.

The explicit patent license in GPLv3 does not go as far as we might have liked. Ideally, we would make everyone who redistributes GPL-covered code give up all software patents, along with everyone who does not redistribute GPL-covered code, because there should be no software patents. Software patents are a vicious and absurd system that puts all software developers in danger of being sued by companies they have never heard of, as well as by all the megacorporations in the field. Large programs typically combine thousands of ideas, so it is no surprise if they implement ideas covered by hundreds

of patents. Megacorporations collect thousands of patents, and use those patents to bully smaller developers. Patents already obstruct free software development.

The only way to make software development safe is to abolish software patents, and we aim to achieve this some day. But we cannot do this through a software license. Any program, free or not, can be killed by a software patent in the hands of an unrelated party, and the program's license cannot prevent that. Only court decisions or changes in patent law can make software development safe from patents. If we tried to do this with GPLv3, it would fail.

Therefore, GPLv3 seeks to limit and channel the danger. In particular, we have tried to save free software from a fate worse than death: to be made effectively proprietary, through patents. The explicit patent license of GPLv3 makes sure companies that use the GPL to give users the four freedoms cannot turn around and use their patents to tell some users, "That doesn't include you." It also stops them from colluding with other patent holders to do this.

Further advantages of GPLv3 include better internationalization, gentler termination, support for BitTorrent, and compatibility with the Apache license. All in all, plenty of reason to upgrade.

Change is unlikely to cease once GPLv3 is released. If new threats to users' freedom develop, we will have to develop GPL version 4. It is important to make sure that programs will have no trouble upgrading to GPLv4 if and when we write one.

One way to do this is to release a program under "GPL version 3 or any later version." Another way is for all the contributors to a program to state a proxy who can decide on upgrading to future GPL versions. The third way is for all the contributors to assign copyright to one designated copyright holder, who will be in a position to upgrade the license version. One way or another, programs should provide this flexibility for future GPL versions.

Chapter 30

The GNU Lesser General Public License

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

- **0. Additional Definitions.**

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

- **1. Exception to Section 3 of the GNU GPL.**

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

- **2. Conveying Modified Versions.**

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a. under this License, provided that you

make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

- b. under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

- **3. Object Code Incorporating Material from Library Header Files.**

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a. Give prominent notice with each copy of the object code that the Library is used in it

and that the Library and its use are covered by this License.

- b. Accompany the object code with a copy of the GNU GPL and this license document.

• **4. Combined Works.**

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a. Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b. Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c. For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library in the

the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

- d. Do one of the following:
 - 0. Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
 - 1. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified

operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

- e. Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

- **5. Combined Libraries.**

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b. Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

• **6. Revised Versions of the GNU Lesser General Public License.**

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such

new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Chapter 31

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

- **0. PREAMBLE**

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by

considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

- **1. APPLICABILITY AND DEFINITIONS**

This License applies to any manual or other work, in any medium that contains a notice placed by

any notices and certain notices placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a

textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, L^AT_EX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or

PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely

of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

- **2. VERBATIM COPYING**

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies

and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

- **3. COPYING IN QUANTITY**

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also

TEXTS ON THE BACK COVER. BOTH COVERS MUST ALSO clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete

Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

- **4. MODIFICATIONS**

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and

modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors,

and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in

Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for

example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

- **5. COMBINING DOCUMENTS**

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

- **6. COLLECTIONS OF DOCUMENTS**

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding

this License in all other respects regarding verbatim copying of that document.

- **7. AGGREGATION WITH INDEPENDENT WORKS**

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form.

covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

- **8. TRANSLATION**

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled

“Acknowledgements,” “Dedications,” or

ACKNOWLEDGEMENTS , Dedications , or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

- **9. TERMINATION**

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright

holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

- **10. FUTURE REVISIONS OF THIS LICENSE**

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See

<http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

• 11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also

server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere

other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) year your name.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no

Invariant Sections, no Front-Cover Texts, and no Back-

Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

with the Invariant Sections being list their titles, with the Front-Cover Texts being list, and with the Back-Cover Texts being list.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Part VI

Traps and Challenges

Chapter 32

Can You Trust Your Computer?

Copyright © 2002, 2007 Richard Stallman

This essay was first published on <http://gnu.org>, in 2002. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Who should your computer take its orders from? Most people think their computers should obey them, not obey someone else. With a plan they call “trusted computing,” large media corporations (including the movie companies and record companies), together with computer companies such as Microsoft and Intel, are planning to make your computer obey them instead of you. (Microsoft’s version of this scheme is called Palladium.) Proprietary programs have included malicious features before, but this plan would make it

malicious features secret, but this plan would make it universal.

Proprietary software means, fundamentally, that you don't control what it does; you can't study the source code, or change it. It's not surprising that clever businessmen find ways to use their control to put you at a disadvantage. Microsoft has done this several times: one version of Windows was designed to report to Microsoft all the software on your hard disk; a recent "security" upgrade in Windows Media Player required users to agree to new restrictions. But Microsoft is not alone: the KaZaA music-sharing software is designed so that KaZaA's business partner can rent out the use of your computer to its clients. These malicious features are often secret, but even once you know about them it is hard to remove them, since you don't have the source code.

In the past, these were isolated incidents. "Trusted computing" would make the practice pervasive. "Treacherous computing" is a more appropriate name, because the plan is designed to make sure your computer will systematically disobey you. In fact, it is designed to stop your computer from functioning as a general-

purpose computer. Every operation may require explicit permission.

The technical idea underlying treacherous computing is that the computer includes a digital encryption and signature device, and the keys are kept secret from you. Proprietary programs will use this device to control which other programs you can run, which documents or data you can access, and what programs you can pass them to. These programs will continually download new authorization rules through the Internet, and impose those rules automatically on your work. If you don't allow your computer to obtain the new rules periodically from the Internet, some capabilities will automatically cease to function.

Of course, Hollywood and the record companies plan to use treacherous computing for Digital Restrictions Management (DRM), so that downloaded videos and music can be played only on one specified computer. Sharing will be entirely impossible, at least using the authorized files that you would get from those companies. You, the public, ought to have both the freedom and the ability to share these things. (I suspect that someone will

ability to share these things. (I expect that someone will find a way to produce unencrypted versions, and to upload and share them, so DRM will not entirely succeed, but that is no excuse for the system.)

Making sharing impossible is bad enough, but it gets worse. There are plans to use the same facility for email and documents—resulting in email that disappears in two weeks, or documents that can only be read on the computers in one company.

Imagine if you get an email from your boss telling you to do something that you think is risky; a month later, when it backfires, you can't use the email to show that the decision was not yours. "Getting it in writing" doesn't protect you when the order is written in disappearing ink.

Imagine if you get an email from your boss stating a policy that is illegal or morally outrageous, such as to shred your company's audit documents, or to allow a dangerous threat to your country to move forward unchecked. Today you can send this to a reporter and expose the activity. With treacherous computing, the reporter won't be able to read the document; her

computer will refuse to obey her. Treacherous computing becomes a paradise for corruption.

Word processors such as Microsoft Word could use treacherous computing when they save your documents, to make sure no competing word processors can read them. Today we must figure out the secrets of Word format by laborious experiments in order to make free word processors read Word documents. If Word encrypts documents using treacherous computing when saving them, the free software community won't have a chance of developing software to read them—and if we could, such programs might even be forbidden by the Digital Millennium Copyright Act.

Programs that use treacherous computing will continually download new authorization rules through the Internet, and impose those rules automatically on your work. If Microsoft, or the US government, does not like what you said in a document you wrote, they could post new instructions telling all computers to refuse to let anyone read that document. Each computer would obey when it downloads the new instructions. Your writing would be subject to 1984-style retroactive censorship. You

would be subject to 1984-style retroactive erasure. You might be unable to read it yourself.

You might think you can find out what nasty things a treacherous-computing application does, study how painful they are, and decide whether to accept them. Even if you can find this out, it would be foolish to accept the deal, but you can't even expect the deal to stand still. Once you come to depend on using the program, you are hooked and they know it; then they can change the deal. Some applications will automatically download upgrades that will do something different—and they won't give you a choice about whether to upgrade.

Today you can avoid being restricted by proprietary software by not using it. If you run GNU/Linux or another free operating system, and if you avoid installing proprietary applications on it, then you are in charge of what your computer does. If a free program has a malicious feature, other developers in the community will take it out, and you can use the corrected version. You can also run free application programs and tools on nonfree operating systems; this falls short of fully giving you freedom, but many users do it.

Treacherous computing puts the existence of free operating systems and free applications at risk, because you may not be able to run them at all. Some versions of treacherous computing would require the operating system to be specifically authorized by a particular company. Free operating systems could not be installed. Some versions of treacherous computing would require every program to be specifically authorized by the operating system developer. You could not run free applications on such a system. If you did figure out how, and told someone, that could be a crime.

There are proposals already for US laws that would require all computers to support treacherous computing, and to prohibit connecting old computers to the Internet. The CBDTPA (we call it the Consume But Don't Try Programming Act) is one of them. But even if they don't legally force you to switch to treacherous computing, the pressure to accept it may be enormous. Today people often use Word format for communication, although this causes several sorts of problems (see "We Can Put an End to Word Attachments," on [\(349\)](#)). If only a

treacherous-computing machine can read the latest Word documents, many people will switch to it, if they view the situation only in terms of individual action (take it or leave it). To oppose treacherous computing, we must join together and confront the situation as a collective choice.

For further information about treacherous computing, see

<http://www.cl.cam.ac.uk/users/rja14/tcpa-faq.html>.

To block treacherous computing will require large numbers of citizens to organize. We need your help! Please support Defective by Design, the FSF's campaign against Digital Restrictions Management.

Postscripts

- 1. The computer security field uses the term “trusted computing” in a different way—beware of confusion between the two meanings.
- 2. The GNU Project distributes the GNU Privacy Guard, a program that implements public-key encryption and digital signatures, which you can

use to send secure and private email. It is useful to explore how GPG differs from treacherous computing, and see what makes one helpful and the other so dangerous.

When someone uses GPG to send you an encrypted document, and you use GPG to decode it, the result is an unencrypted document that you can read, forward, copy, and even reencrypt to send it securely to someone else. A treacherous-computing application would let you read the words on the screen, but would not let you produce an unencrypted document that you could use in other ways. GPG, a free software package, makes security features available to the users; *they* use *it*. Treacherous computing is designed to impose restrictions on the users; *it* uses *them*.

- 3. The supporters of treacherous computing focus their discourse on its beneficial uses. What they say is often correct, just not important. Like most hardware, treacherous-computing hardware can be used for purposes which are not harmful. But these features can be implemented in

other ways, without treacherous-computing hardware. The principal difference that treacherous computing makes for users is the nasty consequence: rigging your computer to work against you.

What they say is true, and what I say is true. Put them together and what do you get? Treacherous computing is a plan to take away our freedom, while offering minor benefits to distract us from what we would lose.

- 4. Microsoft presents Palladium as a security measure, and claims that it will protect against viruses, but this claim is evidently false. A presentation by Microsoft Research in October 2002 stated that one of the specifications of Palladium is that existing operating systems and applications will continue to run; therefore, viruses will continue to be able to do all the things that they can do today.
When Microsoft employees speak of “security” in connection with Palladium, they do not mean what we normally mean by that word: protecting your

machine from things you do not want. They mean protecting your copies of data on your machine from access by you in ways others do not want. A slide in the presentation listed several types of secrets Palladium could be used to keep, including “third party secrets” and “user secrets”—but it put “user secrets” in quotation marks, recognizing that this is somewhat of an absurdity in the context of Palladium.

The presentation made frequent use of other terms that we frequently associate with the context of security, such as “attack,” “malicious code,” “spoofing,” as well as “trusted.” None of them means what it normally means. “Attack” doesn’t mean someone trying to hurt you, it means you trying to copy music. “Malicious code” means code installed by you to do what someone else doesn’t want your machine to do. “Spoofing” doesn’t mean someone’s fooling you, it means your fooling Palladium. And so on.

- 5. A previous statement by the Palladium developers stated the basic premise that whoever

developed or collected information should have total control of how you use it. This would represent a revolutionary overturn of past ideas of ethics and of the legal system, and create an unprecedented system of control. The specific problems of these systems are no accident; they result from the basic goal. It is the goal we must reject.

Chapter 33

Who Does That Server Really Serve?

Copyright © 2010 Richard Stallman

This essay was originally published in the online edition of the *Boston Review*, on 8 March 2010, under the title “What Does That Server Really Serve?” This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Background: How Proprietary Software Takes Away Your Freedom

Digital technology can give you freedom; it can also take your freedom away. The first threat to our control over our computing came from *proprietary software*: software that the users cannot control because the owner (a company such as Apple or Microsoft) controls it. The owner often takes advantage of this unjust power by inserting malicious features such as spyware, back doors,

and Digital Restrictions Management (DRM) (referred to as “Digital Rights Management” in their propaganda).

Our solution to this problem is developing *free software* and rejecting proprietary software. Free software means that you, as a user, have four essential freedoms: (0) to run the program as you wish, (1) to study and change the source code so it does what you wish, (2) to redistribute exact copies, and (3) to redistribute copies of your modified versions. (See “The Free Software Definition,” on [\(9\)](#).)

With free software, we, the users, take back control of our computing. Proprietary software still exists, but we can exclude it from our lives and many of us have done so. However, we now face a new threat to our control over our computing: Software as a Service. For our freedom’s sake, we have to reject that too.

How Software as a Service Takes Away Your Freedom

Software as a Service (SaaS) means that someone sets up a network server that does certain computing tasks—running spreadsheets, word processing, translating text into another language. etc. —then invites users to do their

computing on that server. Users send their data to the server, which does their computing on the data thus provided, then sends the results back or acts on them directly.

These servers wrest control from the users even more inexorably than proprietary software. With proprietary software, users typically get an executable file but not the source code. That makes it hard for programmers to study the code that is running, so it's hard to determine what the program really does, and hard to change it.

With SaaS, the users do not have even the executable file: it is on the server, where the users can't see or touch it. Thus it is impossible for them to ascertain what it really does, and impossible to change it.

Furthermore, SaaS automatically leads to harmful consequences equivalent to the malicious features of certain proprietary software. For instance, some proprietary programs are "spyware": the program sends out data about users' computing activities. Microsoft Windows sends information about users' activities to Microsoft. Windows Media Player and RealPlayer report what each user watches or listens to.

Unlike proprietary software, SaaS does not require covert code to obtain the user's data. Instead, users must send their data to the server in order to use it. This has the same effect as spyware: the server operator gets the data. He gets it with no special effort, by the nature of SaaS.

Some proprietary programs can mistreat users under remote command. For instance, Windows has a back door with which Microsoft can forcibly change any software on the machine. The Amazon Kindle e-book reader (whose name suggests it's intended to burn people's books) has an Orwellian back door that Amazon used in 2009 to remotely delete Kindle copies of Orwell's books *1984* and *Animal Farm* which the users had purchased from Amazon. [\[1\]](#)

SaaS inherently gives the server operator the power to change the software in use, or the users' data being operated on. Once again, no special code is needed to do this.

Thus, SaaS is equivalent to total spyware and a gaping wide back door, and gives the server operator unjust power over the user. We can't accept that.

Untangling the SaaS Issue from the Proprietary Software Issue

SaaS and proprietary software lead to similar harmful results, but the causal mechanisms are different. With proprietary software, the cause is that you have and use a copy which is difficult or illegal to change. With SaaS, the cause is that you use a copy you don't have.

These two issues are often confused, and not only by accident. Web developers use the vague term “web application” to lump the server software together with programs run on your machine in your browser. Some web pages install nontrivial or even large JavaScript programs temporarily into your browser without informing you. When these JavaScript programs are nonfree, they are as bad as any other nonfree software. Here, however, we are concerned with the problem of the server software itself.

Many free software supporters assume that the problem of SaaS will be solved by developing free software for servers. For the server operator's sake, the programs on the server had better be free; if they are nonproprietary, their owners have power over the server

proprietary, their owners have power over the server. That's unfair to the operator, and doesn't help you at all. But if the programs on the server are free, that doesn't protect you *as the server's user* from the effects of SaaS. They give freedom to the operator, but not to you.

Releasing the server software source code does benefit the community: suitably skilled users can set up similar servers, perhaps changing the software. But none of these servers would give you control over computing you do on it, unless it's *your* server. The rest would all be SaaS. SaaS always subjects you to the power of the server operator, and the only remedy is, *Don't use SaaS@!* Don't use someone else's server to do your own computing on data provided by you.

Distinguishing SaaS from Other Network Services

Does condemning SaaS mean rejecting all network servers? Not at all. Most servers do not raise this issue, because the job you do with them isn't your own computing except in a trivial sense.

The original purpose of web servers wasn't to do computing for you, it was to publish information for you

to access. Even today this is what most web sites do, and it doesn't pose the SaaS problem, because accessing someone's published information isn't a matter of doing your own computing. Neither is publishing your own materials via a blog site or a microblogging service such as Twitter or identi.ca. The same goes for communication not meant to be private, such as chat groups. Social networking can extend into SaaS; however, at root it is just a method of communication and publication, not SaaS. If you use the service for minor editing of what you're going to communicate, that is not a significant issue.

Services such as search engines collect data from around the web and let you examine it. Looking through their collection of data isn't your own computing in the usual sense—you didn't provide that collection—so using such a service to search the web is not SaaS. (However, using someone else's search engine to implement a search facility for your own site *is* SaaS.)

E-commerce is not SaaS, because the computing isn't solely yours; rather, it is done jointly for you and another party. So there's no particular reason why you alone should expect to control that computing. The real issue in e-commerce is whether you trust the other party

with your money and personal information.

Using a joint project's servers isn't SaaS because the computing you do in this way isn't yours personally. For instance, if you edit pages on Wikipedia, you are not doing your own computing; rather, you are collaborating in Wikipedia's computing.

Wikipedia controls its own servers, but groups can face the problem of SaaS if they do their group activities on someone else's server. Fortunately, development hosting sites such as Savannah and SourceForge don't pose the SaaS problem, because what groups do there is mainly publication and public communication, rather than their own private computing.

Multiplayer games are a group activity carried out on someone else's server, which makes them SaaS. But where the data involved is just the state of play and the score, the worst wrong the operator might commit is favoritism. You might well ignore that risk, since it seems unlikely and very little is at stake. On the other hand, when the game becomes more than just a game, the issue changes.

Which online services are SaaS@? Google Docs is a

clear example. Its basic activity is editing, and Google encourages people to use it for their own editing; this is SaaS. It offers the added feature of collaborative editing, but adding participants doesn't alter the fact that editing on the server is SaaS. (In addition, Google Docs is unacceptable because it installs a large nonfree JavaScript program into the users' browsers.) If using a service for communication or collaboration requires doing substantial parts of your own computing with it too, that computing is SaaS even if the communication is not.

Some sites offer multiple services, and if one is not SaaS, another may be SaaS. For instance, the main service of Facebook is social networking, and that is not SaaS; however, it supports third-party applications, some of which may be SaaS. Flickr's main service is distributing photos, which is not SaaS, but it also has features for editing photos, which is SaaS.

Some sites whose main service is publication and communication extend it with "contact management": keeping track of people you have relationships with. Sending mail to those people for you is not SaaS, but keeping track of your dealings with them, if substantial, is SaaS.

If a service is not SaaS, that does not mean it is OK. There are other bad things a service can do. For instance, Facebook distributes video in Flash, which pressures users to run nonfree software, and it gives users a misleading impression of privacy. Those are important issues too, but this article's concern is the issue of SaaS.

The IT industry discourages users from considering these distinctions. That's what the buzzword "cloud computing" is for. This term is so nebulous that it could refer to almost any use of the Internet. It includes SaaS and it includes nearly everything else. The term only lends itself to uselessly broad statements.

The real meaning of "cloud computing" is to suggest a devil-may-care approach towards your computing. It says, "Don't ask questions, just trust every business without hesitation. Don't worry about who controls your computing or who holds your data. Don't check for a hook hidden inside our service before you swallow it." In other words, "Think like a sucker." I prefer to avoid the term.

Dealing with the SaaS Problem

Only a small fraction of all web sites do SaaS; most don't raise the issue. But what should we do about the ones that raise it?

For the simple case, where you are doing your own computing on data in your own hands, the solution is simple: use your own copy of a free software application. Do your text editing with your copy of a free text editor such as GNU Emacs or a free word processor. Do your photo editing with your copy of free software such as GIMP.

But what about collaborating with other individuals? It may be hard to do this at present without using a server. If you use one, don't trust a server run by a company. A mere contract as a customer is no protection unless you could detect a breach and could really sue, and the company probably writes its contracts to permit a broad range of abuses. Police can subpoena your data from the company with less basis than required to subpoena them from you, supposing the company doesn't volunteer them like the US phone companies that illegally wiretapped their customers for Bush. If you must use a server, use a server whose operators give you a basis for trust beyond a mere commercial relationship.

However, on a longer time scale, we can create alternatives to using servers. For instance, we can create a peer-to-peer program through which collaborators can share data encrypted. The free software community should develop distributed peer-to-peer replacements for important “web applications.” It may be wise to release them under GNU Affero GPL, since they are likely candidates for being converted into server-based programs by someone else. The GNU Project is looking for volunteers to work on such replacements. We also invite other free software projects to consider this issue in their design.

In the meantime, if a company invites you to use its server to do your own computing tasks, don’t yield; don’t use SaaS. Don’t buy or install “thin clients,” which are simply computers so weak they make you do the real work on a server, unless you’re going to use them with *your* server. Use a real computer and keep your data there. Do your work with your own copy of a free program, for your freedom’s sake.

Endnotes

1 Brad Stone, “Amazon Erases Orwell Books from Kindle,” *New York Times*, 17 July 2009, sec. B1,

<http://nytimes.com/2009/07/18/technology/companies/18amazon.html>.

Chapter 34

Free but Shackled: The Java Trap

Copyright © 2004, 2006, 2010 Richard Stallman

This essay was first published on <http://gnu.org>, in 2004. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Since this article was first published, on 12 April@tie 2004, Sun has relicensed most of its Java platform reference implementation under the GNU General Public License, and there is now a free development environment for Java. Thus, the Java language as such is no longer a trap.

You must be careful, however, because not every Java platform is free. Sun continues distributing an executable Java platform which is nonfree, and other companies do so too.

The free environment for Java is called IcedTea; the source code Sun freed is included in that. So that is the one you should use. Many GNU/Linux distributions come with IcedTea, but some include nonfree Java platforms.

To reliably ensure your Java programs run fine in a free environment, you need to develop them using IcedTea. Theoretically the Java platforms should be compatible, but they are not compatible 100 percent.

In addition, there are nonfree programs with “Java” in their name, such as JavaFX, and there are nonfree Java packages you might find tempting but need to reject. So check the licenses of

whatever packages you plan to use. If you use Swing, make sure to use the free version, which comes with IcedTea.

Aside from those Java specifics, the general issue described here remains important, because any nonfree library or programming platform can cause a similar problem. We must learn a lesson from the history of Java, so we can avoid other traps in the future.

If your program is free software, it is basically ethical—but there is a trap you must be on guard for. Your program, though in itself free, may be restricted by nonfree software that it depends on. Since the problem is most prominent today for Java programs, we call it the Java Trap.

A program is free software if its users have certain crucial freedoms. Roughly speaking, they are: the freedom to run the program, the freedom to study and change the source, the freedom to redistribute the source and binaries, and the freedom to publish improved versions. (See “The Free Software Definition,” on [9](#).) Whether any given program in source form is free software depends solely on the meaning of its license.

Whether the program can be used in the Free World, used by people who mean to live in freedom, is a more complex question. This is not determined by the program’s own license alone, because no program works in isolation. Every program depends on other programs. For instance, a program needs to be compiled or interpreted, so it depends on a compiler or interpreter. If compiled into byte code, it depends on a byte-code interpreter. Moreover, it needs libraries in order to run, and it may also invoke other separate programs that run

in other processes. All of these programs are dependencies. Dependencies may be necessary for the program to run at all, or they may be necessary only for certain features. Either way, all or part of the program cannot operate without the dependencies.

If some of a program's dependencies are nonfree, this means that all or part of the program is unable to run in an entirely free system—it is unusable in the Free World. Sure, we could redistribute the program and have copies on our machines, but that's not much good if it won't run. That program is free software, but it is effectively shackled by its nonfree dependencies.

This problem can occur in any kind of software, in any language. For instance, a free program that only runs on Microsoft Windows is clearly useless in the Free World. But software that runs on GNU/Linux can also be useless if it depends on other nonfree software. In the past, Motif (before we had LessTif) and Qt (before its developers made it free software) were major causes of this problem. Most 3D video cards work fully only with nonfree drivers, which also cause this problem. But the major source of this problem today is Java, because people who write free software often feel Java is sexy. Blinded by their attraction to the language, they overlook the issue of dependencies and fall into the Java Trap.

Sun's implementation of Java is nonfree. The standard Java libraries are nonfree also. We do have free implementations of Java, such as the GNU Compiler for Java (GCJ) and GNU Classpath, but they don't support all the features yet. We are still catching up.

If you develop a Java program on Sun's Java platform, you are liable to use Sun-only features without even noticing. By the time you find this out, you may have been using them for months, and redoing the work could take more months. You might say, "It's too much work to start over." Then your program will have fallen into the Java Trap; it will be unusable in the Free World.

The reliable way to avoid the Java Trap is to have only a free implementation of Java on your system. Then if you use a Java feature or library that free software does not yet support, you will find out straightaway, and you can rewrite that code immediately.

Sun continues to develop additional "standard" Java libraries, and nearly all of them are nonfree; in many cases, even a library's specification is a trade secret, and Sun's latest license for these specifications prohibits release of anything less than a full implementation of the specification. (See

<http://jcp.org/aboutJava/communityprocess/JSPA2.pdf>
and

http://jcp.org/aboutJava/communityprocess/final/jsr129/j2me_pb-1_0-fr-spec-license.html for examples.)

Fortunately, that specification license does permit releasing an implementation as free software; others who receive the library can be allowed to change it and are not required to adhere to the specification. But the requirement has the effect of prohibiting the use of a collaborative development model to produce the free implementation. Use of that model would entail publishing incomplete versions, something those who have read the spec are not allowed to do.

In the early days of the free software movement, it was impossible to avoid depending on nonfree programs. Before we had the GNU C compiler, every C program (free or not) depended on a nonfree C compiler. Before we had the GNU C library, every program depended on a nonfree C library. Before we had Linux, the first free kernel, every program depended on a nonfree kernel. Before we had BASH, every shell script had to be interpreted by a nonfree shell. It was inevitable that our first programs would initially be hampered by these dependencies, but we accepted this because our plan included rescuing them subsequently. Our overall goal, a self-hosting GNU operating system, included free replacements for all those dependencies; if we reached the goal, all our programs would be rescued. Thus it happened: with the GNU/Linux system, we can now run these programs on free platforms.

The situation is different today. We now have powerful free operating systems and many free programming tools. Whatever job you want to do, you can do it on a free platform; there is no need to accept a nonfree dependency even temporarily. The main reason people fall into the trap today is because they are not thinking about it. The easiest solution to the problem is to teach people to recognize it and not fall into it.

To keep your Java code safe from the Java Trap, install a free Java development environment and use it. More generally, whatever language you use, keep your eyes open, and check the free status of programs your code depends on. The easiest way to verify that a program is free is by looking for it in the Free Software Directory (<http://fsf.org/directory>). If a

program is not in the directory, you can check its license(s) against the list of free software licenses (<http://gnu.org/licenses/license-list.html>).

We are trying to rescue the trapped Java programs, so if you like the Java language, we invite you to help in developing GNU Classpath. Trying your programs with the GCJ Compiler and GNU Classpath, and reporting any problems you encounter in classes already implemented, is also useful. However, finishing GNU Classpath will take time; if more nonfree libraries continue to be added, we may never have all the latest ones. So please don't put your free software in shackles. When you write an application program today, write it to run on free facilities from the start.

Chapter 35

The JavaScript Trap

Copyright © 2009, 2010 Richard Stallman

This essay was first published on <http://gnu.org>, in 2009. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

This chapter is licensed under the Creative Commons Attribution-NoDerivs 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

In the free software community, the idea that nonfree programs mistreat their users is familiar. Some of us refuse entirely to install proprietary software, and many others consider nonfreedom a strike against the program. Many users are aware that this issue applies to the plugins that browsers offer to install, since they can be free or nonfree.

But browsers run other nonfree programs which they don't ask you about or even tell you about—programs that web pages contain or link to. These programs are most often written in JavaScript, though other languages are also used.

JavaScript (officially called ECMAScript, but few use that name) was once used for minor frills in web pages, such as cute but inessential navigation and display features. It was acceptable to consider these as mere extensions of HTML markup, rather than as true software; they did not constitute a significant issue.

Many sites still use JavaScript that way, but some use it for major programs that do large jobs. For instance, Google Docs downloads into your machine a JavaScript program which measures half a megabyte, in a compacted form that we could call Obfuscrypt because it has no comments and hardly any whitespace, and the method names are one letter long. The source code of a program is the preferred form for modifying it; the compacted code is not source code, and the real source code of this program is not available to the user.

Browsers don't normally tell you when they load JavaScript programs. Most browsers have a way to turn off JavaScript entirely, but none of them can check for JavaScript programs that are nontrivial and nonfree. Even if you're aware of this issue, it would take you considerable trouble to identify and then block those programs. However, even in the free software community most users are not aware of this issue; the browsers' silence tends to conceal it.

It is possible to release a JavaScript program as free software, by distributing the source code under a free software license. But even if the program's source is available, there is no easy way to run your modified version instead of the original. Current free browsers do not offer a facility to run your own modified version instead of the one delivered in the page. The effect is comparable to tivoization, although not quite so hard to overcome.

JavaScript is not the only language web sites use for programs sent to the user. Flash supports programming through an extended variant of JavaScript. We will need

to study the issue of Flash to make suitable recommendations. Silverlight seems likely to create a problem similar to Flash, except worse, since Microsoft uses it as a platform for nonfree codecs. A free replacement for Silverlight does not do the job for the free world unless it normally comes with free replacement codecs.

Java applets also run in the browser, and raise similar issues. In general, any sort of applet system poses this sort of problem. Having a free execution environment for an applet only brings us far enough to encounter the problem.

A strong movement has developed that calls for web sites to communicate only through formats and protocols that are free (some say “open”); that is to say, whose documentation is published and which anyone is free to implement. With the presence of programs in web pages, that criterion is necessary, but not sufficient. JavaScript itself, as a format, is free, and use of JavaScript in a web site is not necessarily bad. However, as we’ve seen above, it also isn’t necessarily OK. When the site

transmits a program to the user, it is not enough for the program to be written in a documented and unencumbered language; that program must be free, too. “Only free programs transmitted to the user” must become part of the criterion for proper behavior by web sites.

Silently loading and running nonfree programs is one among several issues raised by “web applications.” The term “web application” was designed to disregard the fundamental distinction between software delivered to users and software running on the server. It can refer to a specialized client program running in a browser; it can refer to specialized server software; it can refer to a specialized client program that works hand in hand with specialized server software. The client and server sides raise different ethical issues, even if they are so closely integrated that they arguably form parts of a single program. This article addresses only the issue of the client-side software. We are addressing the server issue separately.

In practical terms, how can we deal with the problem of nonfree JavaScript programs in web sites? Here’s a

of nontrivial JavaScript programs in web sites. Here's a plan of action.

First, we need a practical criterion for nontrivial JavaScript programs. Since “nontrivial” is a matter of degree, this is a matter of designing a simple criterion that gives good results, rather than determining the one correct answer.

Our proposal is to consider a JavaScript program nontrivial if it makes an AJAX request, and consider it nontrivial if it defines methods and either loads an external script or is loaded as one.

At the end of this article we propose a convention by which a nontrivial JavaScript program in a web page can state the URL where its source code is located, and can state its license too, using stylized comments.

Finally, we need to change free browsers to support freedom for users of pages with JavaScript. First of all, browsers should be able to tell the user about nontrivial nonfree JavaScript programs, rather than running them. Perhaps NoScript could be adapted to do this.

Browser users also need a convenient facility to specify JavaScript code to use *instead* of the JavaScript in a certain page. (The specified code might be total replacement, or a modified version of the free JavaScript program in that page.) Greasemonkey comes close to being able to do this, but not quite, since it doesn't guarantee to modify the JavaScript code in a page before that program starts to execute. Using a local proxy works, but is too inconvenient now to be a real solution. We need to construct a solution that is reliable and convenient, as well as sites for sharing changes. The GNU Project would like to recommend sites which are dedicated to free changes only.

These features will make it possible for a JavaScript program included in a web page to be free in a real and practical sense. JavaScript will no longer be a particular obstacle to our freedom—no more than C and Java are now. We will be able to reject and even replace the nonfree nontrivial JavaScript programs, just as we reject and replace nonfree packages that are offered for installation in the usual way. Our campaign for web sites to free their JavaScript can then begin.

Thank you to Matt Lee and John Resig for their help in defining our proposed criterion, and to David Parunakian and Jaffar Rumith for bringing this issue to my attention.

Appendix: A Convention for Releasing Free JavaScript Programs

For references to corresponding source code, we recommend

```
// @source:
```

followed by the URL.

To indicate the license of the JavaScript code embedded in a page, we recommend putting the license notice between two notes of this form:

```
@licstart The following is the entire license notice  
for the JavaScript code in this page.
```

```
...  
@licend The above is the entire license notice  
for the JavaScript code in this page.
```

Of course, all of this should be contained in a multiline comment.

The GNU GPL, like many other free software licenses, requires distribution of a copy of the license with both source and binary forms of the program. However, the GNU GPL is long enough that including it in a page with a JavaScript program can be inconvenient. You can remove that requirement, for code that you have the copyright on, with a license notice like this:

```
Copyright (C) YYYY Developer
```

```
The JavaScript code in this page is free software: you
can redistribute it and/or modify it under the terms of
the GNU General Public License (GNU GPL) as published by
the Free Software Foundation, either version 3 of the
License, or (at your option) any later version. The code
is distributed WITHOUT ANY WARRANTY; without even the
implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU GPL for more details.
```

```
As additional permission under GNU GPL version 3 section
7, you may distribute non-source (e.g., minimized or
compacted) forms of that code without the copy of the GNU
GPL normally required by section 4, provided you include
this license notice and a URL through which recipients
```

this license notice and a URL through which recipients can access the Corresponding Source.

Chapter 36

The X Window System Trap

Copyright © 1998, 1999, 2009 Richard Stallman

This essay was originally published on <http://gnu.org>, in 1998.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

To copyleft or not to copyleft? That is one of the major controversies in the free software community. The idea of copyleft is that we should fight fire with fire—that we should use copyright to make sure our code stays free. The GNU General Public License (GNU GPL) is one example of a copyleft license.

Some free software developers prefer noncopyleft distribution. Noncopyleft licenses such as the XFree86 and BSD licenses are based on the idea of never saving

no to anyone—not even to someone who seeks to use your work as the basis for restricting other people. Noncopyleft licensing does nothing wrong, but it misses the opportunity to actively protect our freedom to change and redistribute software. For that, we need copyleft.

For many years, the X Consortium was the chief opponent of copyleft. It exerted both moral suasion and pressure to discourage free software developers from copylefting their programs. It used moral suasion by suggesting that it is not nice to say no. It used pressure through its rule that copylefted software could not be in the X Distribution.

Why did the X Consortium adopt this policy? It had to do with their conception of success. The X Consortium defined success as popularity—specifically, getting computer companies to use the X Window System. This definition put the computer companies in the driver's seat: whatever they wanted, the X Consortium had to help them get it.

Computer companies normally distribute proprietary

software. They wanted free software developers to donate their work for such use. If they had asked for this directly, people would have laughed. But the X Consortium, fronting for them, could present this request as an unselfish one. "Join us in donating our work to proprietary software developers," they said, suggesting that this is a noble form of self-sacrifice. "Join us in achieving popularity," they said, suggesting that it was not even a sacrifice.

But self-sacrifice is not the issue: tossing away the defense that copyleft provides, which protects the freedom of the whole community, is sacrificing more than yourself. Those who granted the X Consortium's request entrusted the community's future to the goodwill of the X Consortium.

This trust was misplaced. In its last year, the X Consortium made a plan to restrict the forthcoming X11R6.4 release so that it would not be free software. They decided to start saying no, not only to proprietary software developers, but to our community as well.

There is an irony here. If you said yes when the X

Consortium asked you not to use copyleft, you put the X Consortium in a position to license and restrict its version of your program, along with the code for the core of X.

The X Consortium did not carry out this plan. Instead it closed down and transferred X development to the Open Group, whose staff are now carrying out a similar plan. To give them credit, when I asked them to release X11R6.4 under the GNU GPL in parallel with their planned restrictive license, they were willing to consider the idea. (They were firmly against staying with the old X11 distribution terms.) Before they said yes or no to this proposal, it had already failed for another reason: the XFree86 group followed the X Consortium's old policy, and will not accept copylefted software.

In September 1998, several months after X11R6.4 was released with nonfree distribution terms, the Open Group reversed its decision and rereleased it under the same noncopyleft free software license that was used for X11R6.3. Thus, the Open Group therefore eventually did what was right, but that does not alter the general issue.

Even if the X Consortium and the Open Group had never planned to restrict X, someone else could have done it. Noncopylefted software is vulnerable from all directions; it lets anyone make a nonfree version dominant, if he will invest sufficient resources to add significantly important features using proprietary code. Users who choose software based on technical characteristics, rather than on freedom, could easily be lured to the nonfree version for short-term convenience.

The X Consortium and Open Group can no longer exert moral suasion by saying that it is wrong to say no. This will make it easier to decide to copyleft your X-related software.

When you work on the core of X, on programs such as the X server, Xlib, and Xt, there is a practical reason not to use copyleft. The X.org group does an important job for the community in maintaining these programs, and the benefit of copylefting our changes would be less than the harm done by a fork in development. So it is better to work with them, and not copyleft our changes on these programs. Likewise for utilities such as `xset` and `xrdb`,

which are close to the core of X and do not need major improvements. At least we know that the X.org group has a firm commitment to developing these programs as free software.

The issue is different for programs outside the core of X: applications, window managers, and additional libraries and widgets. There is no reason not to copyleft them, and we should copyleft them.

In case anyone feels the pressure exerted by the criteria for inclusion in the X distributions, the GNU Project will undertake to publicize copylefted packages that work with X. If you would like to copyleft something, and you worry that its omission from the X distribution will impede its popularity, please ask us to help.

At the same time, it is better if we do not feel too much need for popularity. When a businessman tempts you with "more popularity," he may try to convince you that his use of your program is crucial to its success. Don't believe it! If your program is good, it will find many users anyway. You don't need to feel desperate for

many users anyway, you don't need to feel desperate for any particular users, and you will be stronger if you do not. You can get an indescribable sense of joy and freedom by responding, "Take it or leave it—that's no skin off my back." Often the businessman will turn around and accept the program with copyleft, once you call the bluff.

Friends, free software developers, don't repeat old mistakes! If we do not copyleft our software, we put its future at the mercy of anyone equipped with more resources than scruples. With copyleft, we can defend freedom, not just for ourselves, but for our whole community.

Chapter 37

The Problem Is Software Controlled by Its Developer

Copyright © 2008, 2010 Richard Stallman

This article was first published in the March/April 2008 issue of <http://bostonreview.net> and is a response to Jonathan Zittrain's "Protecting the Internet without Wrecking It," which was published in the same issue and is available at

<http://bostonreview.net/BR33.2/zittrain.php>. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

I fully agree with Jonathan Zittrain's conclusion that we should not abandon general-purpose computers. Alas, I disagree completely with the path that led him to it. He presents serious security problems as an intolerable crisis, but I'm not convinced. Then he forecasts that

users will panic in response and stampede toward restricted computers (which he calls “appliances”), but there is no sign of this happening.

Zombie machines are a problem, but not a catastrophe. Moreover, far from panicking, most users ignore the issue. Today, people are indeed concerned about the danger of phishing (mail and web pages that solicit personal information for fraud), but using a browsing-only device instead of a general computer won’t protect you from that.

Meanwhile, Apple has reported that 25 percent of iPhones have been unlocked. Surely at least as many users would have preferred an unlocked iPhone but were afraid to try a forbidden recipe to obtain it. This refutes the idea that users generally prefer that their devices be locked.

It is true that a general computer lets you run programs designed to spy on you, restrict you, or even let the developer attack you. Such programs include KaZaA, RealPlayer, Adobe Flash, Windows Media Player, Microsoft Windows, and Mac OS X Windows

Player, MICROSOFT WINDOWS, and MACOS. WINDOWS Vista does all three of those things; it also lets Microsoft change the software without asking, or command it to permanently cease normal functioning.

But restricted computers are no help, because they present the same problem for the same reason.

The iPhone is designed for remote attack by Apple. When Apple remotely destroys iPhones that users have unlocked to enable other uses, that is no better than when Microsoft remotely sabotages Vista. The TiVo is designed to enforce restrictions on access to the recordings you make, and reports what you watch. E-book readers such as the Amazon “Swindle” are designed to stop you from sharing and lending your books. Features that artificially obstruct use of your data are known as Digital Restrictions Management (DRM); our protest campaign against DRM is hosted at <http://defectivebydesign.org>. (Our adversaries call DRM “Digital Rights Management” based on their idea that restricting you is their right. When you choose a term, you choose your side.)

The nastiest of the common restricted devices are cell phones. They transmit signals for tracking your whereabouts even when switched “off”; the only way to stop this is to take out all the batteries. Many can also be turned on remotely, for listening, unbeknownst to you. (The FBI is already taking advantage of this feature, and the US Commerce Department lists this danger in its Security Guide.) Cellular phone network companies regularly install software in users phones, without asking, to impose new usage restrictions.

With a general computer you can escape by rejecting such programs. You don’t have to have KaZaA, RealPlayer, Adobe Flash, Windows Media Player, Microsoft Windows or MacOS on your computer (I don’t). By contrast, a restricted computer gives you no escape from the software built into it.

The root of this problem, both in general PCs and restricted computers, is software controlled by its developer. The developer (typically a corporation) controls what the program does, and prevents everyone else from changing it. If the developer decides to put in malicious features, even a masterless computer cannot

malicious features, even a master programmer cannot easily remove them.

The remedy is to give the users more control, not less. We must insist on free/libre software, software that the users are free to change and redistribute. Free/libre software develops under the control of its users: if they don't like its features, for whatever reason, they can change them. If you're not a programmer, you still get the benefit of control by the users. A programmer can make the improvements you would like, and publish the changed version. Then you can use it too.

With free/libre software, no one has the power to make a malicious feature stick. Since the source code is available to the users, millions of programmers are in a position to spot and remove the malicious feature and release an improved version; surely someone will do it. Others can then compare the two versions to verify independently which version treats users right. As a practical fact, free software is generally free of designed-in malware.

Many people do acquire restricted devices, but not

for motives of security. Why do people choose them?

Sometimes it is because the restricted devices are physically smaller. I edit text all day (literally) and I find the keyboard and screen of a laptop well worth the size and weight. However, people who use computers differently may prefer something that fits in a pocket. In the past, these devices have typically been restricted, but they weren't chosen for that reason.

Now they are becoming less restricted. In fact, the OpenMoko cell phone features a main computer running entirely free/libre software, including the GNU/Linux operating system normally used on PCs and servers.

A major cause for the purchase of some restricted computers is financial sleight of hand. Game consoles, and the iPhone, are sold for an unsustainably low price, and the manufacturers subsequently charge when you use them. Thus, game developers must pay the game console manufacturer to distribute a game, and they pass this cost on to the user. Likewise, AT&T pays Apple when an iPhone is used as a telephone. The low up-front price misleads customers into thinking they will save money

misleads customers into thinking they will save money.

If we are concerned about the spread of restricted computers, we should tackle the issue of the price deception that sells them. If we are concerned about malware, we should insist on free software that gives the users control.

Postnote

Zittrain's suggestion to reduce the statute of limitations on software patent lawsuits is a tiny step in the right direction, but it is much easier to solve the whole problem. Software patents are an unnecessary, artificial danger imposed on all software developers and users in the US. Every program is a combination of many methods and techniques—thousands of them in a large program. If patenting these methods is allowed, then hundreds of those used in a given program are probably patented. (Avoiding them is not feasible; there may be no alternatives, or the alternatives may be patented too.) So the developers of the program face hundreds of potential lawsuits from parties unknown, and the users can be sued as well.

The complete, simple solution is to eliminate patents from the field of software. Since the patent system is created by statute, eliminating patents from software will be easy given sufficient political will. (See <http://www.endsoftpatents.org>.)

Chapter 38

We Can Put an End to Word Attachments

Copyright © 2002, 2007, 2010 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2002.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Don't you just hate receiving Word documents in email messages? Word attachments are annoying, but, worse than that, they impede people from switching to free software. Maybe we can stop this practice with a simple collective effort. All we have to do is ask each person who sends us a Word file to reconsider that way of doing things.

Most computer users use Microsoft Word. That is unfortunate for them, since Word is proprietary software, denying its users the freedom to study, change, copy, and redistribute it. And because Microsoft changes the Word file format with each release, its users are locked into a system that compels them to buy each upgrade whether they want a change or not. They may even find, several years from now, that the Word documents they are writing this year can no longer be read with the version of

Word they use then.

But it hurts us, too, when they assume we use Word and send us (or demand that we send them) documents in Word format. Some people publish or post documents in Word format. Some organizations will only accept files in Word format: I heard from someone that he was unable to apply for a job because resumes had to be Word files. Even governments sometimes impose Word format on the public, which is truly outrageous.

For us users of free operating systems, receiving Word documents is an inconvenience or an obstacle. But the worst impact of sending Word format is on people who might switch to free systems: they hesitate because they feel they must have Word available to read the Word files they receive. The practice of using the secret Word format for interchange impedes the growth of our community and the spread of freedom. While we notice the occasional annoyance of receiving a Word document, this steady and persistent harm to our community usually doesn't come to our attention. But it is happening all the time.

Many GNU users who receive Word documents try to find ways to handle them. You can manage to find the somewhat obfuscated ASCII text in the file by skimming through it. Free software today can read most Word documents, but not all—the format is secret and has not been entirely decoded. Even worse, Microsoft can change it at any time.

Worst of all, it has already done so. Microsoft Office 2007 uses by default a format based on the patented OOXML format. (This is the one that Microsoft got declared an “open standard” by political manipulation and packing standards committees.) The actual format is not entirely OOXML, and it is not entirely documented. Microsoft offers a gratis patent license for OOXML on terms which do not allow free implementations. We are thus beginning to receive Word files in a format that free programs are not even allowed to read.

When you receive a Word file, if you think of that as an isolated event, it is natural to try to cope by finding a way to read it. Considered as an instance of a pernicious systematic practice, it calls for a different approach. Managing to read the file is treating a symptom of an epidemic disease; what we really want to do is stop the disease from spreading. That means we must convince people not to send or post Word documents.

I therefore make a practice of responding to Word attachments with a polite message explaining why the practice of sending Word files is a bad thing, and asking the person to resend the material in a nonsecret format. This is a lot less work than trying to read the somewhat obfuscated ASCII text in the Word file. And I find that people usually understand the issue, and many say they will not send Word files to others any more.

If we all do this, we will have a much larger effect. People who disregard one polite request may change their practice when they receive multiple polite requests from various people. We were able to win Dan's

from various people. We may be able to give *Don't send Word format!* the status of netiquette, if we start systematically raising the issue with everyone who sends us Word files.

To make this effort efficient, you will probably want to develop a canned reply that you can quickly send each time it is necessary. I've included two examples: the version I have been using recently, followed by a new version that teaches a Word user how to convert to other useful formats.

- You sent the attachment in Microsoft Word format, a secret proprietary format, so I cannot read it. If you send me the plain text, HTML, or PDF, then I could read it.

Sending people documents in Word format has bad effects, because that practice puts pressure on them to use Microsoft software. In effect, you become a buttress of the Microsoft monopoly. This specific problem is a major obstacle to the broader adoption of GNU/Linux. Would you please reconsider the use of Word format for communication with other people?

- You sent the attachment in Microsoft Word format, a secret proprietary format, so it is hard for me to read. If you send me plain text, HTML, or PDF, then I will read it.

Distributing documents in Word format is bad for you and for others. You can't be sure what they will look like if someone views them with a different version of Word; they may not work at

all.

Receiving Word documents is bad for you because they can carry viruses (see

[http://en.wikipedia.org/wiki/Macro_virus__\(computing\)](http://en.wikipedia.org/wiki/Macro_virus__(computing))).

Sending Word documents is bad for you because a Word document normally includes hidden information about the author, enabling those in the know to pry into the author's activities (maybe yours). Text that you think you deleted may still be embarrassingly present. See

<http://news.bbc.co.uk/2/hi/technology/3154479.stm> for more info.

But above all, sending people Word documents puts pressure on them to use Microsoft software and helps to deny them any other choice. In effect, you become a buttress of the Microsoft monopoly. This pressure is a major obstacle to the broader adoption of free software. Would you please switch to a different way of sending files to other people, instead of Word format?

To convert the file to HTML using Word is simple. Open the document, click on **File**, then **Save As**, and in the **Save As Type** strip box at the bottom of the box, choose **HTML Document** or **Web Page**. Then choose **Save**. You can then attach the new HTML document instead of your Word document. Note that Word changes in inconsistent ways—if you see slightly different menu item names, please try them.

To convert to plain text is almost the same—instead of HTML Document, choose Text Only or Text Document as the Save As Type.

Your computer may also have a program to convert to PDF format. Select File, then Print. Scroll through available printers and select the PDF converter. Click on the Print button and enter a name for the PDF file when requested.

See <http://gnu.org/philosophy/no-word-attachments.html> for more about this issue.

You can use these replies verbatim if you like, or you can personalize them or write your own. By all means construct a reply that fits your ideas and your personality—if the replies are personal and not all alike, that will make the campaign more effective.

These replies are meant for individuals who send Word files. When you encounter an organization that imposes use of Word format, that calls for a different sort of reply; there you can raise issues of fairness that would not apply to an individual's actions.

Some recruiters ask for resumes in Word format. Ludicrously, some recruiters do this even when looking for someone for a free software job. (Anyone using those recruiters for free software jobs is not likely to get a competent employee.) To help change this practice, you can put a link to <http://gnu.org/philosophy/no-word-attachments.html> into your resume, next to

links to other formats of the resume. Anyone hunting for a Word version of the resume will probably read the page.

This essay talks about Word attachments, since they are by far the most common case. However, the same issues apply with other proprietary formats, such as PowerPoint and Excel. Please feel free to adapt the replies to cover those as well.

With our numbers, simply by asking, we can make a difference.

Chapter 39

Thank You, Larry McVoy

Copyright © 2005 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2005.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

For the first time in my life, I want to thank Larry McVoy. He recently eliminated a major weakness of the free software community, by announcing the end of his campaign to entice free software projects to use and promote his nonfree software. Soon, Linux development will no longer use this program, and no longer spread the message that nonfree software is a good thing if it's convenient.

My gratitude is limited. since it was McVoy that

created the problem in the first place. But I still appreciate his decision to clear it up.

There are thousands of nonfree programs, and most merit no special attention, other than developing a free replacement. What made this program, BitKeeper, infamous and dangerous was its marketing approach: inviting high-profile free software projects to use it, so as to attract other paying users.

McVoy made the program available gratis to free software developers. This did not mean it was free software for them: they were privileged not to part with their money, but they still had to part with their freedom. They gave up the fundamental freedoms that define free software: freedom to run the program as you wish for any purpose, freedom to study and change the source code as you wish, freedom to make and redistribute copies, and freedom to publish modified versions.

The free software movement has said, “Think of ‘free speech,’ not ‘free beer’” since 1990. McVoy said the opposite; he invited developers to focus on the lack of

monetary price, instead of on freedom. A free software activist would dismiss this suggestion, but those in our community who value technical advantage above freedom and community were susceptible to it.

McVoy's great triumph was the adoption of this program for Linux development. No free software project is more visible than Linux. It is the kernel of the GNU/Linux operating system, an essential component, and users often mistake it for the entire system. As McVoy surely planned, the use of his program in Linux development was powerful publicity for it.

It was also, whether intentionally or not, a powerful political PR campaign, telling the free software community that freedom-denying software is acceptable as long as it's convenient. If we had taken that attitude towards Unix in 1984, where would we be today? Nowhere. If we had accepted using Unix, instead of setting out to replace it, nothing like the GNU/Linux system would exist.

Of course, the Linux developers had practical reasons for what they did. I won't argue with those

reasons; they surely know what's convenient for them. But they did not count, or did not value, how this would affect their freedom—or the rest of the community's efforts.

A free kernel, even a whole free operating system, is not sufficient to use your computer in freedom; we need free software for everything else, too. Free applications, free drivers, free BIOS: some of those projects face large obstacles—the need to reverse engineer formats or protocols or pressure companies to document them, or to work around or face down patent threats, or to compete with a network effect. Success will require firmness and determination. A better kernel is desirable, to be sure, but not at the expense of weakening the impetus to liberate the rest of the software world.

When the use of his program became controversial, McVoy responded with distraction. For instance, he promised to release it as free software if the company went out of business. Alas, that does no good as long as the company remains in business. Linux developers responded by saying, “We’ll switch to a free program

when you develop a better one.” This was an indirect way of saying, “We made the mess, but we won’t clean it up.”

Fortunately, not everyone in Linux development considered a nonfree program acceptable, and there was continuing pressure for a free alternative. Finally Andrew Tridgell developed an interoperating free program, so Linux developers would no longer need to use a nonfree program.

McVoy first blustered and threatened, but ultimately chose to go home and take his ball with him: he withdrew permission for gratis use by free software projects, and Linux developers will move to other software. The program they no longer use will remain unethical as long as it is nonfree, but they will no longer promote it, nor by using it teach others to give freedom low priority. We can begin to forget about that program.

We should not forget the lesson we have learned from it: Nonfree programs are dangerous to you and to your community. Don’t let them get a place in your life.

Part VII

An Assessment and a Look Ahead

Chapter 40

Computing “Progress”: Good and Bad

Copyright © 2006, 2007 Richard Stallman

The BBC invited me to write an article for their column series, *The Tech Lab*, and this is what I sent them. (It refers to a couple of other articles published in that series.) The BBC was ultimately unwilling to publish it with a copying-permission notice, so I published it on <http://gnu.org>, in 2007. This version of this essay is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Bradley Horowitz of Yahoo proposed here [\[1\]](#) that every object in our world have a unique number so that your cell phone could record everything you do—even which cans you picked up while in the supermarket.

If the phone is like today's phones. it will use

proprietary software: software controlled by the companies that developed it, not by its users. Those companies will ensure that your phone makes the information it collects about you available to the phone company's database (let's call it Big Brother) and probably to other companies.

In the UK of the future, as New Labour would have it, those companies will surely turn this information over to the police. If your phone reports you bought a wooden stick and a piece of poster board, the phone company's system will deduce that you may be planning a protest, and report you automatically to the police so they can accuse you of "terrorism."

In the UK, it is literally an offense to be suspect—more precisely, to possess any object in circumstances that create a "reasonable suspicion" that you might use it in certain criminal ways. Your phone will give the police plenty of opportunities to suspect you so they can charge you with having been suspected by them. Similar things will happen in China, where Yahoo has already given the government all the information it needed to imprison a

dissident; it subsequently asked for our understanding on the excuse that it was “just following orders.”

Horowitz would like cell phones to tag information automatically, based on knowing when you participate in an event or meeting. That means the phone company will also know precisely whom you meet. That information will also be interesting to governments, such as those of the UK and China, that cut corners on human rights.

I do not much like Horowitz’s vision of total surveillance. Rather, I envision a world in which our computers never collect, or release, any information about us except when we want them to.

Nonfree software does other nasty things besides spying; it often implements digital handcuffs—features designed to restrict the users (also called DRM, for Digital Restrictions Management). These features control how you can access, copy, or move the files in your own computer.

DRM is a common practice: Microsoft does it, Apple does it, Google does it, even the BBC’s iPlayer

does it. Many governments, taking the side of these companies against the public, have made it illegal to tell others how to escape from the digital handcuffs. As a result, competition does nothing to check the practice: no matter how many proprietary alternatives you might have to choose from, they will all handcuff you just the same. If the computer knows where you are located, it can make DRM even worse: there are companies that would like to restrict what you can access based on your present location.

My vision of the world is different. I would like to see a world in which all the software in our computers — in our desktop PCs, our laptops, our handhelds, our phones — is under our control and respects our freedom. In other words, a world where all software is *free* software.

Free software, freedom-respecting software, means that every user of the program is free to get the program's source code and change the program to do what she wants, and also free to give away or sell copies, either exact or modified. This means the users are in

control. With the users in control of the software, nobody has power to impose nasty features on others.

Even if you don't exercise this control yourself, you are part of a society where others do. If you are not a programmer, other users of the program are. They will probably find and remove any nasty features, which might spy on or restrict you, and publish safe versions. You will have only to elect to use them—and since all other users will prefer them, that will usually happen with no effort on your part.

Charles Stross envisioned computers that permanently record everything that we see and hear. [\[2\]](#) Those records could be very useful, as long as Big Brother doesn't see and hear all of them. Today's cell phones are already capable of listening to their users without informing them, at the request of the police, the phone company, or anyone that knows the requisite commands. As long as phones use nonfree software, controlled by its developers and not by the users, we must expect this to get worse. Only free software enables computer-using citizens to resist totalitarian surveillance.

Dave Winer's article [3] suggested that Mr. Gates should send a copy of Windows Vista to Alpha Centauri. I understand the feeling, but sending just one won't solve our problem here on Earth. Windows is designed to spy on users and restrict them. We should collect all the copies of Windows, and of MacOS and iPlayer for the same reason, and send them to Alpha Centauri at the slowest possible speed. Or just erase them.

Endnotes

1 Bradley Horowitz, "The Tech Lab: Bradley Horowitz," *BBC News*, 29 June 2007,
<http://news.bbc.co.uk/2/hi/technology/6252716.stm>.

2 Charles Stross, "The Tech Lab: Charles Stross," *BBC News*, 10 July 2007,
<http://news.bbc.co.uk/2/hi/technology/6287126.stm>.

3 Dave Winer, "The Tech Lab: Dave Winer," *BBC News*, 14 June 2007,
<http://news.bbc.co.uk/2/hi/technology/6748103.stm>.

Chapter 41

Avoiding Ruinous Compromises

Copyright © 2008, 2009 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2008.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

This chapter is licensed under the Creative Commons Attribution-NoDerivs 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

The free software movement aims for a social change: to make all software free so that all software users are free and can be part of a community of cooperation. Every nonfree program gives its developer unjust power over the users. Our goal is to put an end to that injustice.

The road to freedom is a long road. It will take many

steps and many years to reach a world in which it is normal for software users to have freedom. Some of these steps are hard, and require sacrifice. Some of them become easier if we make compromises with people that have different goals.

Thus, the Free Software Foundation makes compromises—even major ones. For instance, we made compromises in the patent provisions of version 3 of the GNU General Public License (GNU GPL) so that major companies would contribute to and distribute GPLv3-covered software and thus bring some patents under the effect of these provisions.

The Lesser GPL's purpose is a compromise: we use it on certain chosen free libraries to permit their use in nonfree programs because we think that legally prohibiting this would only drive developers to proprietary libraries instead. We accept and install code in GNU programs to make them work together with common nonfree programs, and we document and publicize this in ways that encourage users of the latter to install the former, but not vice versa. We support specific

campaigns we agree with, even when we don't fully agree with the groups behind them.

But we reject certain compromises even though many others in our community are willing to make them. For instance, we endorse only the GNU/Linux distributions that have policies not to include nonfree software or lead users to install it. To endorse nonfree distributions would be a ruinous compromise.

Compromises are ruinous if they would work against our aims in the long term. That can occur either at the level of ideas or at the level of actions.

At the level of ideas, ruinous compromises are those that reinforce the premises we seek to change. Our goal is a world in which software users are free, but as yet most computer users do not even recognize freedom as an issue. They have taken up “consumer” values, which means they judge any program only on practical characteristics such as price and convenience.

Dale Carnegie's classic self-help book, *How to Win Friends and Influence People*, advises that the most

effective way to persuade someone to do something is to present arguments that appeal to his values. There are ways we can appeal to the consumer values typical in our society. For instance, free software obtained gratis can save the user money. Many free programs are convenient and reliable, too. Citing those practical benefits has succeeded in persuading many users to adopt various free programs, some of which are now quite successful.

If getting more people to use some free programs is as far as you aim to go, you might decide to keep quiet about the concept of freedom, and focus only on the practical advantages that make sense in terms of consumer values. That's what the term "open source" and its associated rhetoric do.

That approach can get us only part way to the goal of freedom. People who use free software only because it is convenient will stick with it only as long as it is convenient. And they will see no reason not to use convenient proprietary programs along with it.

The philosophy of open source presupposes and appeals to consumer values, and this affirms and

appeals to consumer values, and this admits and reinforces them. That's why we do not support open source.

To establish a free community fully and lastingly, we need to do more than get people to use some free software. We need to spread the idea of judging software (and other things) on “citizen values,” based on whether it respects users’ freedom and community, not just in terms of convenience. Then people will not fall into the trap of a proprietary program baited by an attractive, convenient feature.

To promote citizen values, we have to talk about them and show how they are the basis of our actions. We must reject the Dale Carnegie compromise that would influence their actions by endorsing their consumer values.

This is not to say we cannot cite practical advantage at all—we can and we do. It becomes a problem only when the practical advantage steals the scene and pushes freedom into the background. Therefore, when we cite the practical advantages of free software, we reiterate

frequently that those are just *additional, secondary* reasons to prefer it.

It's not enough to make our words accord with our ideals; our actions have to accord with them too. So we must also avoid compromises that involve doing or legitimizing the things we aim to stamp out.

For instance, experience shows that you can attract some users to GNU/Linux if you include some nonfree programs. This could mean a cute nonfree application that will catch some user's eye, or a nonfree programming platform such as Java (formerly) or the Flash runtime (still), or a nonfree device driver that enables support for certain hardware models.

These compromises are tempting, but they undermine the goal. If you distribute nonfree software, or steer people towards it, you will find it hard to say, "Nonfree software is an injustice, a social problem, and we must put an end to it." And even if you do continue to say those words, your actions will undermine them.

The issue here is not whether people should be *able*

or *allowed* to install nonfree software; a general-purpose system enables and allows users to do whatever they wish. The issue is whether we guide users towards nonfree software. What they do on their own is their responsibility; what we do for them, and what we direct them towards, is ours. We must not direct the users towards proprietary software as if it were a solution, because proprietary software is the problem.

A ruinous compromise is not just a bad influence on others. It can distort your own values, too, through cognitive dissonance. If you have certain values, but your actions imply other, conflicting values, you are likely to change your values or your actions so as to resolve the contradiction. Thus, projects that argue only from practical advantages, or direct people toward some nonfree software, nearly always shy away from even *suggesting* that nonfree software is unethical. For their participants, as well as for the public, they reinforce consumer values. We must reject these compromises if we wish to keep our values straight.

If you want to move to free software without

compromising the goal of freedom, look at the FSF's resources area. It lists hardware and machine configurations that work with free software, totally free GNU/Linux distros to install, and thousands of free software packages that work in a 100 percent free software environment. If you want to help the community stay on the road to freedom, one important way is to publicly uphold citizen values. When people are discussing what is good or bad, or what to do, cite the values of freedom and community and argue from them.

A road that lets you go faster is no improvement if it leads to the wrong place. Compromise is essential to achieve an ambitious goal, but beware of compromises that lead away from the goal.

Chapter 42

Overcoming Social Inertia

Copyright © 2007, 2009 Richard Stallman

This essay was originally published on <http://gnu.org>, in 2007.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Almost two decades have passed since the combination of GNU and Linux first made it possible to use a PC in freedom. We have come a long way since then. Now you can even buy a laptop with GNU/Linux preinstalled from more than one hardware vendor—although the systems they ship are not entirely free software. So what holds us back from total success?

The main obstacle to the triumph of software freedom is social inertia. It exists in many forms, and you

have surely seen some of them. Examples include devices that only work on Windows and commercial web sites accessible only with Windows. If you value short-term convenience instead of freedom, you might consider these reason enough to use Windows. Most companies currently run Windows, so students who think short-term want to learn how to use it and ask their schools to teach it. Schools teach Windows, produce graduates that are used to using Windows, and this encourages businesses to use Windows.

Microsoft actively nurtures this inertia: it encourages schools to inculcate dependency on Windows, and contracts to set up web sites that then turn out to work only with Internet Explorer.

A few years ago, Microsoft ads argued that Windows was cheaper to run than GNU/Linux. Their comparisons were debunked, but it is worth noting the deeper flaw in their argument, the implicit premise which cites a form of social inertia: "Currently, more technical people know Windows than GNU/Linux." People who value their freedom would not give it up to save money,

but many business executives believe ideologically that everything they possess, even their freedom, should be for sale.

Social inertia consists of people who have given in to social inertia. When you surrender to social inertia, you become part of the pressure it exerts on others; when you resist it, you reduce it. We conquer social inertia by identifying it, and resolving not to be part of it.

Here a weakness holds our community back: most GNU/Linux users have never even heard the ideas of freedom that motivated the development of GNU, so they still judge matters based on short-term convenience rather than on their freedom. This makes them vulnerable to being led by the nose by social inertia, so that they become part of the inertia.

To build our community's strength to resist, we need to talk about free software and freedom—not merely about the practical benefits that open source supporters cite. As more people recognize what they need to do to overcome the inertia, we will make more progress.

Chapter 43

Freedom or Power?

Copyright © 2001, 2009 Bradley M. Kuhn and Richard Stallman

This essay was originally published on <http://gnu.org>, in 2001.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

Written by Bradley M. Kuhn and Richard Stallman.

The love of liberty is the love of others; the love of power is the love of ourselves.

—William Hazlitt

In the free software movement, we stand for freedom for the users of software. We formulated our views by looking at what freedoms are necessary for a good way

of life, and permit useful programs to foster a community of goodwill, cooperation, and collaboration. Our criteria for free software specify the freedoms that a program's users need so that they can cooperate in a community.

We stand for freedom for programmers as well as for other users. Most of us are programmers, and we want freedom for ourselves as well as for you. But each of us uses software written by others, and we want freedom when using that software, not just when using our own code. We stand for freedom for all users, whether they program often, occasionally, or not at all.

However, one so-called freedom that we do not advocate is the “freedom to choose any license you want for software you write.” We reject this because it is really a form of power, not a freedom.

This oft overlooked distinction is crucial. Freedom is being able to make decisions that affect mainly you; power is being able to make decisions that affect others more than you. If we confuse power with freedom, we will fail to uphold real freedom.

Making a program proprietary is an exercise of power. Copyright law today grants software developers that power, so they and only they choose the rules to impose on everyone else—a relatively small number of people make the basic software decisions for all users, typically by denying their freedom. When users lack the freedoms that define free software, they can't tell what the software is doing, can't check for back doors, can't monitor possible viruses and worms, can't find out what personal information is being reported (or stop the reports, even if they do find out). If it breaks, they can't fix it; they have to wait for the developer to exercise its power to do so. If it simply isn't quite what they need, they are stuck with it. They can't help each other improve it.

Proprietary software developers are often businesses. We in the free software movement are not opposed to business, but we have seen what happens when a software business has the “freedom” to impose arbitrary rules on the users of software. Microsoft is an egregious example of how denying users' freedoms can lead to direct harm, but it is not the only example. Even

when there is no monopoly, proprietary software harms society. A choice of masters is not freedom.

Discussions of rights and rules for software have often concentrated on the interests of programmers alone. Few people in the world program regularly, and fewer still are owners of proprietary software businesses. But the entire developed world now needs and uses software, so software developers now control the way it lives, does business, communicates, and is entertained. The ethical and political issues are not addressed by the slogan of “freedom of choice (for developers only).”

If “code is law,” [\[1\]](#) then the real question we face is: who should control the code you use—you, or an elite few? We believe you are entitled to control the software you use, and giving you that control is the goal of free software.

We believe you should decide what to do with the software you use; however, that is not what today’s law says. Current copyright law places us in the position of power over users of our code, whether we like it or not. The ethical response to this situation is to proclaim

The ethical response to this situation is to proclaim freedom for each user, just as the Bill of Rights was supposed to exercise government power by guaranteeing each citizen's freedoms. That is what the GNU General Public License is for: it puts you in control of your usage of the software while protecting you from others who would like to take control of your decisions.

As more and more users realize that code is law, and come to feel that they too deserve freedom, they will see the importance of the freedoms we stand for, just as more and more users have come to appreciate the practical value of the free software we have developed.

Endnotes

1 William J. Mitchell, *City of Bits: Space, Place, and the Infobahn* (Cambridge, Mass.: MIT Press, 1995), p. 111, as quoted by Lawrence Lessig in *Code and Other Laws of Cyberspace, Version 2.0* (New York, NY: Basic Books, 2006), p. 5.

Part VIII

Appendices

Appendix A

A Note on Software

This note was originally published in 2002, in the first edition.

This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

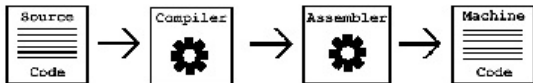
Written by Richard E. Buckman and Joshua Gay.

This section is intended for people who have little or no knowledge of the technical aspects of computer science. It is not necessary to read this section to understand the essays and speeches presented in this book; however, it may be helpful to those readers not familiar with some of the jargon that comes with programming and computer science.

A computer *programmer* writes software, or computer programs. A program is more or less a recipe with *commands* to tell the computer what to do in order

to carry out certain tasks. You are more than likely familiar with many different programs: your Web browser, your word processor, your email client, and the like.

A program usually starts out as *source code*. This higher-level set of commands is written in a *programming language* such as C or Java. After that, a tool known as a *compiler* translates this to a lower-level language known as *assembly language*. Another tool known as an *assembler* breaks the assembly code down to the final stage of *machine language*—the lowest level—which the computer understands *natively*.



For example, consider the “hello world” program, a common first program for people learning C, which (when compiled and executed) prints “Hello World!” on the screen. [\[1\]](#)

```
int main() {  
    printf("Hello World!");  
    return 0;  
}
```

```
}
```

In the Java programming language the same program would be written like this:

```
public class hello {  
    public static void main(String args[]) {  
        System.out.println("Hello World!");  
    }  
}
```

However, in machine language, a small section of it may look similar to this:

```
110001111011101010010100100100101010101110  
011010101001100000111100101101010111101  
0100111111111110010110110000000010100100  
0100100001100101011011000110110001101111  
0010000001010111011011110111001001101100  
0110010000100001010000100110111101101111
```

The above form of machine language is the most basic representation known as binary. All data in computers is made up of a series of 0-or-1 values, but a person would have much difficulty understanding the data. To make a simple change to the binary, one would have to have an intimate knowledge of how a particular computer interprets the machine language. This could be feasible for small programs like the above examples, but

reasonable for small programs like the above examples, but any interesting program would involve an exhausting effort to make simple changes.

As an example, imagine that we wanted to make a change to our “Hello World” program written in C so that instead of printing “Hello World” in English it prints it in French. The change would be simple; here is the new program:

```
int main() {  
    printf("Bonjour, monde!");  
    return 0;  
}
```

It is safe to say that one can easily infer how to change the program written in the Java programming language in the same way. However, even many programmers would not know where to begin if they wanted to change the binary representation. When we say “source code,” we do not mean machine language that only computers can understand—we are speaking of higher-level languages such as C and Java. A few other popular programming languages are C++, Perl, and Python. Some are harder than others to understand and program in, but they are all much easier to work with compared to the intricate machine language they get

compared to the intricate machine language they get turned into after the programs are compiled and assembled.

Another important concept is understanding what an *operating system* is. An operating system is the software that handles input and output, memory allocation, and task scheduling. Generally one considers common or useful programs such as the *Graphical User Interface* (GUI) to be a part of the operating system. The GNU/Linux operating system contains a both GNU and non-GNU software, and a *kernel* called *Linux*. The kernel handles low-level tasks that applications depend upon such as input/output and task scheduling. The GNU software comprises much of the rest of the operating system, including GCC, a general-purpose compiler for many languages; GNU Emacs, an extensible text editor with many, many features; GNOME, the GNU desktop; GNU libc, a library that all programs other than the kernel must use in order to communicate with the kernel; and Bash, the GNU command interpreter that reads your command lines. Many of these programs were pioneered by Richard Stallman early on in the GNU Project and come with any modern GNU/Linux operating system.

It is important to understand that even if *you* cannot

change the source code for a given program, or directly use all these tools, it is relatively easy to find someone who can. Therefore, by having the source code to a program you are usually given the power to change, fix, customize, and learn about a program—this is a power that you do not have if you are not given the source code. Source code is one of the requirements that makes a piece of software *free*. The other requirements will be found along with the philosophy and ideas behind them in this collection.

Endnotes

[1](#) In other programming languages, such as Scheme, the *Hello World* program is usually not your first program. In Scheme you often start with a program like this:

```
(define (factorial n)
  (if (= n 0)
      1
```

```
      (* n (factorial (- n 1)))))
```

This computes the factorial of a number; that is, running `(factorial 5)` would output 120, which is computed by doing $5 * 4 * 3 * 2 * 1 * 1$.

Appendix B

Translations of the Term

“Free Software”

The most current list of translations is maintained at <http://www.gnu.org/philosophy/fs-translations.html>. Please e-mail any additional translations to web-translators@gnu.org. This version of the list is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

Verbatim copying and distribution of this entire chapter are permitted worldwide, without royalty, in any medium, provided this notice is preserved.

The following is a list of recommended unambiguous translations of the term “free software” into various languages:

- Afrikaans: vrye sagteware
- Albanian: software i lirë
- Arabic: برمجيات حرة
- Belarusian: свабоднае праграмнае

забесьпячэньне

- Bulgarian: свободен софтуер
- Catalan: programari lliure
- Chinese: 自由软件 (simplified), 自由軟體 (traditional)
- Czech: svobodný software
- Croatian/Serbian: slobodni softver
- Danish: fri software *or* frit programmel
- Dutch: vrije software
- Esperanto: libera programaro
- Estonian: vaba tarkvara
- Farsi: نرم افزار آزاد
- Finnish: vapaa ohjelmisto
- French: logiciel libre
- German: freie Software
- Greek: ελεύθερο λογισμικό
- Hungarian: szabad szoftver
- Icelandic: frjáls hugbúnaður
- Ido: libera programaro
- Indonesian: perangkat lunak bebas
- Interlingua: libere programmation *or* libere programmario
- Irish: bog earraí saoire
- Italian: software libero

- Japanese: □□□□□□□□ (□□□□□□□□)
- Lithuanian: laisva programinė įranga
- Malay: perisian bebas
- Norwegian: fri programvare
- Polish: wolne oprogramowanie
- Portuguese: software livre
- Romanian: software liber
- Russian: свободное программное обеспечение
- Sardinian: software liberu
- Serbian/Croatian: слободни софтвер
- Slovak: slobodný softvér
- Slovenian: prosto programje
- Spanish: software libre
- Swahili: Programu huru za Kompyuta
- Swedish: fri programvara, fri mjukvara
- Tagalog: malayang software
- Tamil: கட்டற்ற மென்பொருள்
- Turkish: özgür yazılım
- Ukrainian: вільне програмне забезпечення
- Welsh: meddalwedd rydd
- Zulu: Isoftware Ekhululekile